64 PLUS 4





LIPIEC/SIERPIEŃ 1991 ISSN 0867-3918 INDEKS 377112 CENA 10.000 zł

MIESIĘCZNIK UŻYTKOWNIKÓW KOMPUTERÓW COMMODORE

ABUK

Tanio sprzedam COMMODORE 64 II + magnetofon + joysticki + literatura + ok. 400 programów. Informacje: Artur Karaźniewicz, 11-111 Kraszewo, woj. olsztyńskie.

Kupię AMIGĘ 500, sprzedam ATARI 130XE, XCA12, oprogramowanię i literaturę. 34-511 Kościelisko 976.

Sprzedam: C64, monitor, joysticki,cartridges, magnetofon, programy - 2.000.000 zł. Włocławek, tel. 62-316.

Wymienię się chętnie grami lub kupię je (C-64, magnetofon). Kubyszek Bartosz, ul. Tuwima 5/11, 43-300 Bielsko Biała, tel. 454-62.

C-64. Sprzedaż oprogramowania. Bardzo tanio. Nowości !!! Andrzej Tchórzewski, ul. Warszawska 2/22,19-303 Ełk.

Public Domain C-64, C-128(D). Dysk 15 tys., taśma 16 tys. Lesław Pasoń, Rynek 10/1, 42-600 Tarnowskie Góry.

Sprzedam C-64-C, stację dysków i monitor za 3.000.000 zł. Wschowa, tel. 30-68.

C-64+stacja 1541II+magnetofon; wymienię literaturę i oprogramowanie. Tomasz Zarzycki, 59-255 Tomaszów Bol. 130, woj. legnickie.

C-64 II, magnet. 2 x joystick + przedłużacze, cartridge, 15 kaset z programami. Gwarancja! Ok. 3.300tys. Jelenia Góra, Karłowicza 2/164, Niklewicz.

Klub korespondencyjny THE BLOODY GREMLINS zacznie wymieniać oprogramowanie z AMIGOWCAMI. Adres: The Bloody Gremlins - Wójcicki Kamil, ul. Nowogrodzka 60, 95-200 Pabianice, tel. 15-02-60.

POSZUKUJĘ PROGRAMÓW 3D NA AMIGĘ. Tel. 520-335, ul. Kościuszki 112/4, Poznań.

DIGITIZER DŹWIĘKU do A500, cena około 200tys. zł. Informacje (koperta+znaczek): M.P.S. ul. Rąbieńska 24, 94-227 Łódź.

OGŁOSZENIA

KOMPUTEROWA FIRMA USŁUGOWA "TREND" - COMMODORE AMIGA: oprogramowanie, literatura. Informacja : dyskietka lub koperta + znaczek. Kontakt: Rafał Wierzbicki, ul. Rogowska 86/10, 54-440 Wrocław.

THE MANY AMI nawiąże kontakt z amigowcami. Kuba Kopczyk, Os. Wichrowe Wzg. 26/73, 61-697 Poznań.

64 PLUS 4

miesięcz. nr 7/8(9/10) lipiec/sierpień '91
cena 1 egz.: 10.000 zł



Wydawca:
ABUK Spółka z o. o.

Redakcja nie ponosi odpowiedzialności za treść ogłoszeń.

Adres redakcji: Redakcja „64 plus 4" 85-166 Bydgoszcz 43 skrytka pocztowa 64
Redagują: Marcin Dudar, SamborKuźma, Paweł Sołtysiński, Waldemar Szczygieł (red. nacz.), Robert Turliński, Wojciech Wasilewski.
Okładka: Sławomir Karolczak.
Skład: ABUK
Druk: Z.P. POLRASTER
85-353 Bydgoszcz, ul. Orawska 19

OD REDAKCJI

Wadliwa dystrybucja „64 plus 4 & Amiga" przez przedsiębiorstwo RUCH jest przyczyną kłopotów, jakie mają czytelnicy chcący nabyć nasze pismo. Zdarza się, że otrzymujemy jako zwroty NIETKNIĘTE paczki zbiorcze. Tymczasem czytelnicy sygnalizują, że do wielu kiosków nie dociera ono wcale. Dlatego:

zapraszamy wszystkich chętnych do prowadzenia kolportażu „64 plus 4 & Amiga" (kluby, studia i sklepy komputerowe, księgarnie, osoby indywidualne itd.) do współpracy!

Oferujemy korzystne warunki!

Zainteresowanych prosimy o kontakt z działem dystrybucji pod adresem: Przedsiębiorstwo ABUK, 87-200 Wąbrzeźno, ul. 1 Maja 33.

NOWOŚĆ!

PUBLIC DOMAIN PACK dla COMMODORE 64 NA KASECIE!

Wystarczy wpłacić na nasze konto 30.000 zł (wykorzystując blankiet wpłaty zamieszczony wewnątrz numeru), aby stać się posiadaczem kasety zawierającej wybór programów, które do tej pory ukazały się na dyskowych zestawach PDP, a które posiadają wersję taśmową.

Przedsiębiorstwo ABUK S-ka z o.o. oferuje państwu **szybką i tanią obsługę reklamową.** Ogłoszenia drobne od osób indywidualnych (do 10 słów) przyjmujemy bezpłatnie. Większe - 1000 zł za słowo. Reklamy ramkowe (minimalny format - 20 cm^2): 1cm^2 ogłoszenia-4500zł, cała strona - 2,5 mln zł; kolor - odpowiednio 100% drożej.

Ogłoszenia przyjmujemy za pośrednictwem poczty (nasz adres - patrz stopka redakcyjna). Treść ogłoszenia z określeniem formatu reklamy (ewentualnie zamówieniem koloru) prosimy nadsyłać listem poleconym wraz z odcinkiem wpłaty (za pomocą przekazu pieniężnego) na konto Przedsiębiorstwa ABUK Bank Polska Kasa Opieki SA Oddział w Bydgoszczy, konto nr : 5.09011-400522.7-136-11-111.0 Dołączenie do zamówienia odcinka wpłaty przyspieszy zamieszczenie reklamy.

W numerze :

W następnym numerze :

- **Przedstawimy gry: Z-OUT, FUTURE SHOCK i inne**
- **Własne demo na Amidze c.d.**
- **itd, itp.**

Z daleka i z bliska

- Tak, to możliwe! Karta 68030 (22 MHz lub 33 MHz) z koprocesorem MC68882, do tego rozszerzenie pamięci 32-Bit-RAM i kontroler SCSI. To wszystko na JEDNEJ karcie do A2000 oferuje firma GVP. Wersja karty 22 MHz może zawierać do 13MB 32-Bit-RAM, wersja 33 MHz daje możliwość korzystania z maksymalnie 16MB. Standardowo wersja 22 MHz jest wyposażona w 1MB 32-Bit-RAM, 33 MHz w 4MB. Kontroler SCSI jest połączony z szyną 68030, co umożliwia szybsze przenoszenie danych (w porównaniu do poprzednich wersji o ok. 150 KB/s). Karta może zostać przełączana software'owo i hardware'owo w tryb 68000, lecz nie ma wtedy dostępu do pamięci i kontrolera. Nic innego, tylko kupować.

- Firma Rossmoeller przedstawiła stacje dysków HD do Amigi. "Mega Drive"pracuje z dyskietkami 3.5" i ma pojemność do 1.52 MB. Standard 880 KB jest również czytany przez stacje, co daje możliwość pracy ze zwykłymi dyskietkami. Stosunkowo interesująca jest cena: ok. 300 DM. Dwa lata temu 200DM kosztowała dodatkowa, standardowa stacja...

- Wszystkie typy Amig mogą posiadać teraz 2 MB Chip-RAM. Cena tej przyjemności waha się w okolicach 300 DM.

- Za około 50 DM można zakupić interfejs pozwalający Amidze na korzystanie z klawiatury potentata na rynku polskim - komputera IBM AT.

- ACD zaprezentowała... kartę 68040 do A2000 o nazwie "Fusion-40"amerykańskiej firmy RCS. Dla przypomnienia: MC68040 jest następcą MC68030 mikoprocesora MC68882 - zawiera w sobie oba te układy. W 68040 znajduje się m.in. 4 KB Cache (w MC 68030 tylko 256 B), co znakomicie skraca czas dostępu. Ze względu na wbudowanie koprocesora "do środka" uzyskano przyspieszenie o czynnik do 10 razy. MC68040 pracuje wewnętrznie z 50 MHz, chociaż zewnętrznie jest traktowany tylko 25 MHz. Karta "Fusion-40" to naprawde nic wielkiego: 18MIPS (to jest 3-5 razy szybciej niż A3000), możliwość rozszerzenia do maksymalnie 32 MB RAM. Porównanie prędkości przy pomocy "CPU-Speed" daje czynnik przyspieszenia do 27.4 razy. Dla porównania: z kartą 68030 50MHz czynnik ten wynosi 19.8. Cena "Fusion-40" wraz z 4 MB RAM nie jest wysoka, zważywszy na możliwości - ok. 6000. Serce się kraje, gdy człowiek zagląda do portfela.

- Oprócz CD-ROM naszą Amigę można wyposażyć w magneto-optyczną (zapisywalną!) stację dysków Ricoh MOx-600. Jak sama nazwa wskazuje, na dyskietkę (czy raczej dysk) da się wpisać 600 MB danych. Ceny stacji, jak rownież jednej dyskietki, nie podaję - po co się denerwować (8950 DM).

- Amiga jest naprawde dziecinnym komputerem. Dlaczego zatem do tej zabawki tylko jeden sklep oferuje: ok. 10 typów stacji dysków, 8 typów emulatorów IBM, 35 typów kart przyspieszających (tzw. "dopalaczy"), ok. 15 typów kontrolerów dysków twardych, napędów nie zliczę? Powariowały te firmy czy co? Pchać się w tak niepewny biznes...

Jarosław Chrostowski



MAPA PAMIĘCI CZĘŚĆ VII

ADRES HEX	ADRES DEC	ETYKIETA	OPIS
07CD-07D8	1997-2008		obszar roboczy RS 232(ACIA)
07D1	2001	INQFPT	wskaźnik początku bufora wejściowego
07D2	2002	INQRPT	wskaźnik końca bufora wejściowego
07D3	2003	INQCNT	ilość znaków w buforze wejściowym
07D4	2004	ASTAT	chwilowy status ACIA
07D8	2008	APRES	wskaźnik: oznacza obecność ACIA
07D9-07E4	2009-2020	KLUDES	podprogram ładowania znaku
07E5	2021	SCBOT	okno ekranu: dolna krawędź
07E6	2022	SCTOP	okno ekranu: górna krawędź
07E7	2023	SCLF	okno ekranu: lewa krawędź
07E8	2024	SCRT	okno ekranu: prawa krawędź
07E9	2025	SCRDIS	0 = scrolling włączony
07EA	2026	INSFLG	wskaźnik: automatyczne wstawianie
07EB	2027	LSTCHR	ostatni wyprowadzony znak
07EC	2028	LOGSCR	pamięć dla zarządzania ekranem
07ED	2029	TCOLOR	rejestr chwilowy koloru przy INST i DEL
07EE-07F1	2030-2033	BITABL	tabela łączenia wierszy dla ekranu
07F2-07F5	2034-2037		przechowują zawartość rejestrów procesora przy wykonywaniu SYS
07F2	2034	SAREG	akumulator
07F3	2035	SXREG	rejestr X
07F4	2036	SYREG	rejestr Y
07F5	2037	SPREG	rejestr SP
07F6	2038	LSTX	aktualnie naciśnięty klawisz CHR$(n), $40=żaden
07F7	2039	STPDSB	blokada CTRL-S ($FF=nie działa)
07F8	2040	RAMROM	przełączenia RAM/ROM dla monitora języka maszynowego ($00=ROM, $80=RAM)
07F9	2041	COLSW	przełączenie RAM/ROM dla tabeli barwa/jasność ($00=ROM, $80=RAM)
07FA	2042	FFRMSK	maska ROM dla podzielonego ekranu
07FB	2043	VMBMSK	maska video-RAM podzielonego ekranu
07FC	2044	LSEM	sterowanie silnika magnetofonu
07FD	2045	PALCNT	licznik pomocniczy dla zegara czasu rzeczywistego w systemie PAL
07FE-07FF	2046-2047		nie używane
0800-0BE7	2048-3047	TEDATR	pamięć kolorów (tryb tekstowy)
0C00-0FE7	3072-4071	TEDSCN	pamięć obrazu (tryb tekstowy)
1000-3FFF	4096-16383	BASBGN	BASIC-RAM przy C 16 (tryb tekstowy)
1000-FCFF	4096-64767	BASBGN	BASIC-RAM przy PLUS/4 (tryb tekstowy)
1800-1BE7	6144-7143	TEDATR	pamięć jasności (tryb graficzny)
1C00-1FE7	7168-8167	TEDSCN	pamięć kolorów (tryb graficzny)
2000-3F3F	8192-16191	GRBASE	pamięć obrazu (tryb graficzny)
1000-17FF	4096-6143	BASBGN	BASIC-RAM przy C 16 (tryb graficzny)
4000-FCFF	16384-64767	BASBGN	BASIC-RAM przy PLUS/4(tryb graficzny)
8000-CFFF	32768-53247		interpreter BASIC
D000-D3FF	53248-54271		generator znaków (duże/graficzne)
D400-D7FF	54272-55295		generator znaków (duże/małe)
D800-FBFF	55296-64511		system operacyjny (KERNAL)
FD00-FD0F	64768-64783		ACIA (port szeregowy, tylko Plus/4)
FD10-FD1F	64784-64799		port równoległy (tylko Plus/4)
FDD0-FDDF	64976-64991		moduł bank port
FE00-FEFF	65024-65279		DMA system dyskowy
FF00-FF1F	65280-65311		rejestry układu TED pełniącego min. funkcję sterownika obrazu, dźwięku i obsługującego klawiaturę
FF3E	65342		ROM - select
FF3F	65343		RAM - select
FF40-FFFF	65344-65535		tablice skoków systemu operacyjnego

Zgodnie z zapowiedzią publikujemy dziś ostatnią część mapy pamięci komputerów Commodore C-16, C-116, Plus/4.

Tych, którzy nie posiadają poprzednich numerów naszego pisma informujemy, że cykl obejmował artykuły zamieszczane w numerach od grudnia 1990 roku do numeru bieżącego (z wyjątkiem nr. 5/91 w którym publikowaliśmy artykuł pt. GRZEBANIE W BAJTACH - będący znakomitym uzupełnieniem opisu mapy pamięci). Numery zaległe można nabyć za pośrednictwem redakcji.

Wojciech Wasilewski

Uczymy się grać

Po przerwie wracamy do dalszego rozpoznawania możliwości dźwiękowych komputerów C16/116 i Plus4.

Przedstawiamy program narzędziowy pozwalający tworzyć własne melodie, zapisywać je (taśma/dysk) i odczytywać. Przy przenoszeniu do programu utworów istniejących (z partytur) należy postępować wg zasad zawartych w artykule UCZYMY SIĘ GRAĆ, opublikowanym w numerze z lutego 1991r. naszego miesięcznika.

Przystępując do pracy ustalamy wielkość tablicy. W wierszu 50 i 440 listingu podano wymiar tablicy dla modyfikowanego utworu. Pierwszą pozycją tablicy jest ilość nut, drugą czas trwania dźwięku. W omawianym przykładzie przyjęliśmy wartość 31, bowiem w pierwszych taktach piosenki „Lato" wymienionej we wspomnianym artykule, taka ilość nut występuje w linii „DATA". Na początek proponujemy naszą muzyczną zabawę z tą właśnie piosenką.

Przy innym wymiarze tablicy trzeba odpowiednio zmienić wartość w pętlach (wiersze 70, 150, 350 i 550) naszego listingu.

W tej fazie nauki muzykowania używamy tylko generatora dźwięku nr 1. W następnych artykułach przedstawimy bardziej skomplikowane sposoby posługiwania się dwoma generatorami, wraz z „szumami", a także sposoby wbudowywania ilustracji muzycznych i dźwiękowych do własnych programów.

UWAGA:

- odstępy 1 spacji występujące w wierszach z PRINT oraz INPUT, są zwykłymi odstępami pomiędzy wyrazami. Większą ilość odstępów oznaczamy następująco: np. (2sp) - 2 spacje.
- 2 down - oznacza dwukrotne naciśnięcie klawisza kursora ze strzałką „w dół".
- 2 right - j.w. lecz klawisza ze strzałką w prawo.

Życzymy Wam przyjemnej zabawy i wielu estetycznych wrażeń

A oto nasz program.

```
1   REM
2   REM TWORZENIE, ZAPIS I ODCZYT MELODII
3   REM
10  SCNCLR
20  PRINTSPC(122) "BEDZIEMY TWORZYC COS
    NOWEGO, CZY OBRA-"
30  PRINT"(2sp)BIAC WCZESNIEJSZA TWORCZOSC
    (N/W)?":GETKEYA$
40  IFA$="W"THEN420
50  DIMM(31,2)
60  SCNCLR
70  FORJ=0TO30
80  G=1:REM * GENERATOR NR.1 *
90  PRINTJ+1;:INPUT"WARTOSC DZWIEKU (6sp)-";D
100 PRINTJ+1;:INPUT"CZAS TRWANIA DZWIEKU-";C
110 M(J,0)=G
120 M(J,1)=D
130 M(J,2)=C
140 NEXTJ
150 FORJ=0TO30
160 VOL3
170 SOUNDM(J,0),M(J,1),M(J,2)
180 NEXTJ
190 PRINTTAB(2)"(2down) ZAPISUJEMY TO, CZY
    PRACUJEMY DALEJ(7sp)(Z/P)? (K-KONIEC)"
    :GETKEYA$
200 IFA$="K"THEN   SCNCLR:GOTO600
210 IFA$="P"THEN   SCNCLR:GOTO70
220 REM
230 REM * ZAPIS OPRACOWANEJ MELODII *
240 REM
250 PRINT"(2sp)MAGNETOFON, CZY STACJA DYS
    KOW(M/D)?":GETKEYA$
260 IFA$="M"THEN320
270 REM
280 REM * STACJA DYSKOW *
290 REM
300 OPEN1,8,3,"MELODIA,S,W":GOTO350
310 REM
320 REM * MAGNETOFON *
330 REM
340 OPEN1,1,1,"MELODIA"
350 FORJ=0TO30
360 PRINT#1,M(J,0)
370 PRINT#1,M(J,1)
380 PRINT#1,M(J,2)
390 NEXTJ
400 CLOSE1:GOTO600
410 REM
420 REM ** ODCZYT ZAPISANEJ MELODII **
430 REM
440 DIMM(31,2)
450 PRINT"(2sp)Z MAGNETOFONU, CZY ZE STACJI
    DYSKOW   (6sp)(M/D)?"
460 GETKEYA$:IFA$="M"THEN520
470 REM
480 REM ** STACJA DYSKOW **
490 REM
500 OPEN1,8,3,"MELODIA,S,R":GOTO550
510 REM
520 REM ** MAGNETOFON **
530 REM
540 OPEN1,1,0,"MELODIA"
550 FORJ=0TO30
560 INPUT#1,M(J,0),M(J,1),M(J,2)
570 NEXTJ
580 CLOSE1
590 GOTO150
600 SCNCLR: PRINT"(12down,12right) DZIEKUJE-
    CZESC!":END
```

Opracował na podstawie „64'ER", sonderheft nr 8/86

Jan Siedlecki

Graj aby wygrać

EXOLON

Aby stać się nieśmiertelnym wystarczy wejść do opcji definiowania klawiszy i wpisać hasło ZORBA.

SHORT CIRCUIT

Przejście do drugiego poziomu odbywa się po naciśnięciu klawiszy *, ↑, =, RETURN.

CYBERNOID II

Zdefiniowanie klawiszy: YRGO - da nam nieśmiertelność.

POKE 12987,0 - **broń**
SYS 35564

GIANA SISTERS

Aby przejść do następnego poziomu wystarczy nacisnąć jednocześnie klawisze ARMIN.

POKE 8257 - **życie**
POKE 2447,x - **x-ilość żyć**
POKE 2128,x - **x-komnata startowa**
SYS 2127

DRUID

POKE 34068,173 - **energia**
POKE 35787,233
POKE 35788,0 - **energia w wodzie**
POKE 37095,0 - **broń**
SYS 12200
lub
SYS 5120

Uwaga! Posiadacze przystawek FINAL muszą dodatkowo wykonać rozkaz KILL.

GHOSTS'N'S GOBLINS

POKE 2358,173 - **życie**
POKE 11770,0 - **kolizja**
SYS 2128

CLIFF HANGER

POKE 5528,0 - **życie**
POKE 2,94
SYS 3072

PIR²

POKE 14882 - **życie**
POKE 4253,0
POKE 4254,0 - **energia**
SYS 4096

BATTLE ZONE

POKE 39513,165
SYS 49152

Miłej zabawy życzy

S.K.

JAK ZROBIĆ DEMO (4)

Klika wydań C64+4 wcześniej przedstawiłem procedurę służącą obsłudze tzw. scroll'a jednoznakowego (czy, jak kto woli - „płynącego napisu"). Na dokładny opis zasady działania brakło miejsca a i do tekstu programu wkradł się błąd. Postanowiłem więc ponownie zaprezentować, jak prosta może być realizacja tego typu scroll'i - dodatkowo wzbogaconych o możliwość zmiany prędkości płynącego napisu. Podam przykłady dwóch takich procedur: pierwsza dla liter o rozmiarach 8x8 punktów (czyli np. standardowe litery z ROM), a druga dla liter 16x16 (czyli 2 znaki na 2 znaki).

Zacznijmy od napisania procedury otwierającej przerwania i ustawiającej parametry:

```
start JSR $E518  ;standardowe ustawienie VIC-a
      LDX #$27
      LDA #$01
loop  STA $D800,x ;ustawienie koloru białego
                  dla dwóch górnych linii
      DEX
      BPL loop
      SEI
      LDA #$7F
      STA $DC0D
      LDX #$00
      STX $DC0E
      INX
      STX $D01A
      LDA #$1B
      STA $D011
      LDA #$31
      STA $D012  ;przerwanie ustalono na linię
                  o numerze $31
      LDA #irq
      STA $0314
      LDA #irq
      STA $0315  ;ustawienie wektora przerwań
                  na etykietę IRQ
      JSR init   ;wywołanie procedury
                  startującej scroll
      CLI        ;włączanie przerwań
      RTS        ;i powrót
```

Procedura ta będzie pasowała tak do pierwszego, jak i do drugiego scroll'a.

Załóżmy, że tekst, wyświetlany na ekranie, zapisany będzie w pamięci pod postacią kodów ekranowych (kod ASCII litery AND #$3F). Stosując taką technikę łatwo zauważyć, że do zapisu liter i podstawowych znaków wystarczą kody od 1 do 63, co oznacza, że w każdym bajcie tekstu 2 najstarsze bity ($40 i $80) pozostaną nie użyte. My wykorzystamy je do informowania procedury scroll'a o zmianie prędkości tekstu i pauzie. Przyjmijmy zatem, że kody od $41 to $48 będą wyznaczały prędkość tekstu, a kod o wartości $50 (litera 'p') będzie oznaczać pauzę. Zero tradycyjnie będzie oznaczać koniec tekstu i spowoduje jego ponowne zapętlenie.

SCROLL nr 1 (1 x 1):

```
irq   LDA w lewo
      STA $D016  ;rejestr VIC'a - poziomy scroll
                  sprzętowy
      LDA #$3B
loop2 CMP $D012  ;odczekanie czasu
                  ekranowego (1 wiersz)
      BNE loop2
      LDA #$C8
      STA $D016  ;ustawienie wartości
                  standardowej
      JSR scroll ;wywołanie scrolla
      INC $D019  ;potwierdzenie następnego
                  przerwania
      JMP $EA31  ;skok do standardowego IRQ
                  w ROM
init  LDA #$00
      STA pauza  ;ustawienie pauzy w pozycji
                  „wyłączona"
      LDA #$01
      STA speed  ;ustawienie prędkości na 1
      LDA #$07
      STA w lewo ;ustawienie parametru scroll'a
                  w lewo
settext LDA #<tekst
      STA $FB
      LDA #>tekst
      STA $FC    ;ustawienie początku tekstu
                  w $FB/$FC
      RTS
speed            .BYTE 1
pauza            .BYTE 0
w lewo           .BYTE 7
scroll LDA pauza
      BEQ niema  ;nie ma aktywnej pauzy
                  - skocz dalej
      DEC pauza
      RTS
niema LDA wlewo
      SEC
      SBC speed
      AND #$07
      STA wlewo
      BCC dalej
      RTS
dalej LDY #$00
      LDA ($FB),y
      CMP #$50   ;czy to rozkaz pauzy?
      BNE dalej2 ;nie - idź dalej
  LDA #$80
  STA pauza      ;ustawienie pauzy na 128 ramek
      LDA wlewo
      CLC
      ADC speed
      AND #$07
```

```
         STA wlewo
         JMP incr
dalej2   PHA
         LDX #$00
transf   LDA $0401,x
         STA $0400,x
         INX
         CPX #$27
         BNE transf
         PLA
         BNE dalej3   ;koniec tekstu? nie - skocz dalej
         JSR settext
         JMP dalej
dalej3   CMP #$40     ;czy to kod prędkości?
         BCC dalej4   ;nie - idź dalej
         AND #$07
         STA speed
         JSR incr
         JMP dalej
dalej4   STA $0427    ;to litera - wstaw ją na ekran
         JSR incr     ;ustaw wektor tekstu na
                      następny znak
         RTS          ;powrót
incr     INC $FB      ;procedura do zwiększania
                      wektora $FB/$FC
         BNE end
         INC $FC
end      RTS
```

A oto przykład tekstu:

```
tskst    .BYTE $03, $36, $37, $2B, $34, $20
         .BYTE $42
         .BYTE $03, $36, $37, $2B, $34, $20
         .BYTE $43
         .BYTE $03, $36, $37, $2B, $34, $20
         .BYTE $45
         .BYTE $03, $36, $37, $2B, $34, $20
         .BYTE $50, $41, $00
```

W następnym artykule opiszę procedurę do zrealizowania drugiego z wymienionych scroll'i.

Paweł Sołtysiński (Polonus/Padua)

Programy ciekawe, zwariowane i takie sobie

No nareszcie!!! W końcu uratowaliście mnie i przyszło kilka listów z propozycjami do tej zwariowanej rubryki. Czekam dalej na Wasze propozycje!

Najpierw program Bartosza Polendra ze Szczecina. Program Bartosza jest tzw. lupą do spite'ów. Sprawdźcie sami jak fajnie działa!

```
5 REM (W) BARTOSZ POLENDER
6 :
10 WS=192: REM NUMER DUSZKA (0-255)
20 AD=WS*64: FOR I=1 TO 21
30 FOR J=1 TO 3
40 B=128: C=PEEK (AD)
50 FOR K=1 TO 8
60 IF C>=B THEN PRINT „*";: C=C-B: GOTO 80
70 PRINT „.";
80 B=B/2: NEXT: AD=AD+1: NEXT
```

Autorem drugiego programu jest Rafał Zimowski z Łodzi. Jak pisze w swoim liście jego programik ani nie jest ciekawy, ani zwariowany, ale na pewno taki sobie. Kto zgadnie co jest efektem jego działania?

```
10 FOR I=0 TO 1000
20 P=1+G
30 PRINT CHR$(147): PRINT 0+P
40 G=1+P
50 NEXT
```

S.K.

ASSEMBLER 6510 - lekcja 3

Cześć! Dziś zaczynamy omówienie rozkazów mikroprocesora. W wielu książkach (np. „Commodore 64" Bohdana Frelka) rozkazy te są bardzo dobrze opisane. Zapytacie więc po co o tym chcę pisać? Po pierwsze: nie wszyscy (jak wynika z Waszych listów) posiadają odpowiednie publikacje, po drugie: często najdokładniejszy opis nie daje tyłu wiadomości jak prosty przykład w gotowej procedurze. Mam nadzieję, że zamieszczone przykłady pozwolą Wam pełniej zrozumieć zasady programowania w języku maszynowym.

ROZKAZY MIKROPROCESORA cz.1

UWAGA. Obok rozkazów podany będzie wpływ operacji na rejestr znaczników. Gwiazdka (*) oznacza zmianę bitu (oczywiście w zależności od wyniku) a minus(-) oznacza brak wpływu działania rozkazu na jego stan. Oprócz tego podane zostaną kod każdego rozkazu i ilość cykli. Czasami przy przekraczaniu strony pamięci liczba cykli zwiększa się o jeden. W takim przypadku po plusie dopisana będzie jedynka.

Rozkazy opisane będą tematycznie, dzięki temu łatwiej będzie je można sobie przyswoić. Do dzieła!

LDA (load accumulator) - ładowanie akumulatora

N	V	D	I	Z	C
*	-	-	-	*	-

ROZKAZ	KOD	CYKLE
LDA #$nn	A9	2
LDA $nn	A5	3
LDA $nn,X	B5	4
LDA $nnnn	AD	4
LDA $nnnn,X	BD	4+1
LDA $nnnn,Y	B9	4+1
LDA ($nn,X)	A1	8
LDA ($nn),Y	B1	5+1

Rozkaz ten powoduje załadowanie danej do akumulatora np.

```
1000 LDA #$3E        ; załadowanie 3E
                     do akumulatora
1002 BRK             ; koniec
```

LDX (load X) - ładowanie rejestru X

N	V	D	I	Z	C
*	-	-	-	*	-

ROZKAZ	KOD	CYKLE
LDX #$nn	A2	2
LDX $nn	A6	3
LDX $nn,Y	B6	4
LDX $nnnn	AE	4
LDX $nnnn,Y	BE	4+1

Rozkaz ten działa tak samo jak poprzedni tyle, że dana jest ładowana do rejestru X.

LDY (load Y) - ładowanie rejestru Y

N	V	D	I	Z	C
*	-	-	-	*	-

ROZKAZ	KOD	CYKLE
LDY #$nn	A0	2
LDY $nn	A4	3
LDY $nn,X	B4	4
LDY $nnnn	AC	4
LDY $nnnn,X	BC	4+1

STA (store accumulator) - zapisz akumulator

N	V	D	I	Z	C
-	-	-	-	-	-

ROZKAZ	KOD	CYKLE
STA $nn	85	3
STA $nn,X	95	4
STA $nnnn	8D	4
STA $nnnn,X	9D	5
STA $nnnn,Y	99	5
STA ($nn,X)	81	6
STA ($nn),Y	91	6

Rozkaz ten zapamiętuje dane zawarte w akumulatorze we wskazanej komórce pamięci, np.

```
1000 LDA #$FF        ; ładowanie wartości
                     FF do akumulatora
1002 STA $FB         ; zapisanie jej w komórce
                     o adresie FB
1004 BRK             ; koniec
```

STX (store X) - zapisz rejestr X

N	V	D	I	Z	C
-	-	-	-	-	-

ROZKAZ	KOD	CYKLE
STX $nn	86	3
STX $nn,Y	96	4
STX $nnnn	8E	4

STX zapisuje dane zawarte w rejestrze X pod wskazanym adresem.

STY (store Y) - zapisz Y

N	V	D	I	Z	C
-	-	-	-	-	-

ROZKAZ	KOD	CEKLE
STY $nn	84	3
STY $nn,X	94	4
STY $nnnn	8C	4

Działanie takie samo jak poprzedniego rozkazu tyle, że dotyczy rejestru Y

ADC (add with carry) - dodanie z „carry"

N	V	D	I	Z	C
*	*	-	-	*	*

ROZKAZ	KOD	CYKLE
ADC #$nn	69	2
ADC $nn	65	3
ADC $nn,X	75	4
ADC $nnnn	6D	4+1
ADC $nnnn,X	7D	4+1
ADC $nnnn,Y	79	4
ADC ($nn,X)	61	6
ADC ($nn),Y	71	5+1

Jak sama nazwa mówi rozkaz ADC dodaje do siebie wartości. Co jednak oznacza „z carry"? Wykonajmy taki program:

```
1000 LDA #$10     :ładowanie do akumulatora
                  wartości 10
1002 CLC          ; zerowanie znacznika „C"
                  (carry)
1003 ADC #$1A     ; dodanie wartości 1A
1005 BRK          ; koniec programu
```

W akumulatorze znalazła sie wartość 2A. Zamieńcie teraz rozkaz CLC na SEC (ustawienie znacznika „C"). W wyniku otrzymacie wartość 2B! Rozkaz ADC dodaje do sumy dwóch liczb zawartość 'carry'. Jeśli więc w czasie dodawania nie jesteście pewni zawartości znacznika C trzeba koniecznie go wyzerować (rozkaz CLC).

SBC (subtract with carry) - odejmowanie z carry

N	V	D	I	Z	C
*	*	-	-	*	*

ROZKAZ	KOD	CYKLE
SBC #$nn	E9	2
SBC $nn	E5	3
SBC $nn,X	F5	4
SBC $nnnn	ED	4
SBC $nnnn,X	FD	4+1
SBC $nnnn,Y	F9	4+1
SBC ($nn,X)	E1	6
SBC ($nn),Y	F1	5+1

Rozkaz powoduje odjęcie od siebie dwu wartości; zawartość 'carry' znowu ma wpływ na wynik. Tym razem wartość 'C' jest negowana (zmienia swój stan na przeciwny). Wykonajmy:

```
1000 LDA #$2A     ; załadowanie do
                  akumulatora wartości 2A
1002 SEC          ; ustawienie carry (C=1)
1003 SBC #$1A     ; odjęcie 1A
1005 BRK          ; koniec
```

W wyniku (w akumulatorze) otrzymamy wartość 10. Jeśli zmienimy rozkaz SEC (ustawienie carry) na CLC (skasowanie carry) to wynikiem będzie liczba 0F.

Być może zastanawiacie się po co komu znajomość ilości cykli zajmowanych przez każdy rozkaz? Wartości tych oczywiście wcale nie trzeba uczyć się na pamięć - byłoby to pozbawione sensu. Czasami jednak przy programowaniu skomplikowanych procedur opierających się na ścisłych wyliczeniach czasowych warto mieć te dane. Niekiedy zmiana choćby JEDNEGO! cyklu w procedurze powoduje jej błędne działanie! Dotyczy to zwłaszcza operacji na VIC'u.

Miesiąc temu mieliście zastanowić się nad napisaniem programu relokującego więcej niż $FF (256) bajtów pamięci. Oto jedno z możliwych rozwiązań (dla wszystkich leniuszków, którym nie chciało się samemu pomęczyć):

```
1000    LDA  #$00
        LDX  #$04
        STA  $FB
        STA  $FC
        STA  $FD
        STX  $FE
100C    LDY  #$00
100E    LDA  ($FB),Y
        STA ($FD),Y
        INY
        BNE $100E
        INC $FC
        INC $FE
        LDA $FE
        CMP #$08
        BNE $100C
        BRK
```

Przeanalizujcie ten programik. Jeśli nie znacie któregoś z rozkazów to... zapraszam za miesiąc - wszystko stanie się jasne!

Sambor Kuźma

Disk Checker 64

Zdarzyło mi się niedawno dostać większą ilość dyskietek od przyjaciela, który - jak sam to określił - „już wyrósł" z komputerów. Dyskietek było dużo, zapisy na nich nie odświeżane od lat. Wprawdzie „darowanemu koniowi w zęby się nie zagląda", ale chciałem jednak wiedzieć, które z dyskietek się na coś przydadzą, a których należy się po prostu pozbyć. W tym celu powstał program służący do testowania dyskietek. Sprawdza on je tylko pobieżnie, ale za to szybko i na jego „opinii" można polegać.

W dużym uproszczeniu program nakazuje stacji dysków dotrzeć do każdego miejsca na dyskietce, sprawdzając, czy owo „docieranie" odbyło się bez kłopotów. Jeśli nie było problemów - dyskietka jest raczej w porządku.

Program pracuje w BASIC'u z wykorzystaniem prostej procedury w języku maszynowym. Po wpisaniu do komputera należy go oczywiście - dobrym zwyczajem - nagrać na dyskietkę.

Po uruchomieniu DISK CHECKER'a na ekranie oprócz informacji o numerze aktualnie testowanego track'a, wyświetlona jest tabela, w której w sposób graficzny oznaczono sprawność poszczególnych track'ów (1-35). Kolorem białym oznaczone są track'i sprawne, czarnym - wadliwe.

Hmm... bardzo ciekawym doświadczeniem może być sprawdzenie wszystkich swoich dysków...

Paweł Sołtysiński

```
0 POKE53280,12:POKE53281,12:PRINTCHR$(142)CHR$(5)CHR$(8):GOSUB 100
1 PRINTCHR$(147)CHR$(17)"----------------------------------------";
2 PRINT"DISK CHECKER V.1":A$=CHR$(153):PRINT:POKE832,0
3 PRINT:PRINT:PRINTA$"   0 1 2 3 4 5 6 7 8 9"
4 PRINT"00:X . . . . . . . . ."
5 PRINT"10:. . . . . . . . . ."
6 PRINT"20:. . . . . . . . . ."
7 PRINT"30:. . . . . . X X X X"CHR$(5)
30 OPEN15,8,15:OPEN2,8,2,"#"
35 PRINT#15,"M-W"CHR$(106)CHR$(0)CHR$(1)CHR$(133)
36 FORY=1TO35:PRINTCHR$(19)"TRACK TO CHECK:";Y
40 PRINT#15,"U1:"2;0;Y; :SYS20480
70 NEXTY:CLOSE2:CLOSE15
80 PRINT:PRINT:PRINT"SPACE [illegible]AGAIN; ANY OTHER KEY - EXIT"
90 GETA$:IFA$=""THEN90
91 IFA$=" "THEN RUN1
92 END
100 AD=20480
101 READ C:IF C=-1THEN RETURN
102 POKEAD,C:AD=AD+1:GOTO 101
110 DATA169,13,32,198,255,169,8,32,180,255,169,111,32,150,255,32
111 DATA165,255,174,64,3,188,47,80,201,48,208,3,169,1,252,169
112 DATA0,153,0,217,169,81,153,0,5,238,64,3,76,171,255,29
113 DATA31,33,35,37,39,41,43,45,67,69,71,73,75,77,79,81
114 DATA83,85,107,109,1[illegible]3,115,117,119,121,123,125,147,149,151,153,155,157
115 DATA-1
```

LISTING PROGRAMU

O ZABEZPIECZANIU

Już na wstępie muszę zaznaczyć, że ten artykuł tematycznie przeznaczony jest raczej dla Czytelników piszących własne programy (mam tu na myśli tych spośród Was, którzy dość dobrze znają już system operacyjny C64), które chcieliby dobrze zabezpieczyć lub, po prostu chcieliby coś na ten temat wiedzieć na potrzeby (nie daj Boże, he,he) „łamania" programów innych.

W artykule zajmiemy się raczej problemem, jak utrudnić włamanie niż jak uniemożliwić skopiowanie. W obecnym czasie każdy „szanujący się" hacker nie poprzestaje na uczynieniu programu kopiowalnym, ale dodaje także własne udoskonalenia, jak np. nieśmiertelność, wyłączenie limitu czasu w grach, itp. Osobiście jestem przekonany, że proces zabezpieczania jakiegokolwiek programu jest niewspółmiernie bardziej pracochłonny od techniki odbezpieczania go przez zawodowego hackera. Jedyną pociechą jest fakt, że dobre blokady sprawiają na ogół to, że 99% niedoszłych „łamaczy" odpada jeszcze w przedbiegach. Jak znam praktykę, to w rezultacie programista po skończeniu pracy nad zabezpieczaniem swojego dzieła na ogół może wytypować parę znanych sobie osób, które poradzą sobie z jego blokadami (tak to jakoś jest, że wszyscy się znają...). A więc przystąpmy do rzeczy.

Po bliższym zastanowieniu łatwo stwierdzić, że większość blokad można podzielić na dwa rodzaje:

A - zabezpieczenia, które maksymalnie utrudniają „wdarcie się" do programu, znalezienie jego początku, rozpoznanie, co właściwie dana procedura robi, itp.

B - zabezpieczenia, które mają jakiegoś „haka", tzn. że np. są w stanie wykryć posiadanie przez nas cardridge'a, fałszywego dysku i itp.

Zajmijmy się na razie typem A, do którego należy zaliczyć:

AUTOSTART - czyli wszelkiej maści tzw. loadery, które powodują automatyczne wgranie i uruchomienie interesującego nas programu. Najczęściej spotykane to te, które ładują się z dysku w obszar wektorów systemowych (od $0300), albo jeszcze lepiej - jeszcze „niżej", np. od adresu $0200 lub nawet na stos. Ciekawa jest zwłaszcza ta ostatnia metoda. Jak wiadomo, stos procesora leży od $0100 do $01FF. Wystarczy napisać krótką procedurę ładującą począwszy od adresu $0102 a całą resztę stosu wypełnić wartością $01. Teraz podczas najbliższego rozkazu RTS procesor skoczy do naszej procedury (czyli pod $0102). Należy jednak obiektywnie stwierdzić, że dla prawdziwego hackera nie jest to wcale wielka trudność i na ogół spotyka się z komentarzem w stylu „eee tam! zawracanie głowy...".

ZAMIANA WEKTORÓW - jeśli tylko nasz program nie korzysta z procedur w pamięci ROM, lub korzysta tylko z niektórych, to wektory tych nieużywanych stanowią wdzięczne pole do popisu dla naszej wyobraźni. I tak możemy na przykład ustawić wektory LOAD i SAVE na adres procedury RESET w pamięci ROM (adres $FCE2), zablokować RUN/STOP-RESTORE, napisać procedurę kasującą całą pamięć i skierować na nią wektor trybu edycji BASIC'a ($0302/$0303). Jeśli jeszcze nam się chce coś jeszcze zmieniać, to możemy dowolnie pomodyfikować wektory określające położenie w pamięci programu w języku BASIC, itp. W praktyce dużo hałasu a mało efektu...

SKOKI „PRZEZ STOS" - czyli nic innego jak upchanie na stosie dwóch wartości w formie starszego i młodszego bajtu adresu skoku pomniejszonego o jeden i wykonanie rozkazu RTS. Wszystko jasne i wszyscy wiedzą jak to działa, ale zapewniam, że tak około 150 takich sztuczek i każdy hacker padnie z wywieszonym językiem przed swoim komputerem. Najbardziej wskazane jest umieszczanie (o ile to możliwe) tej sztuczki przy każdej nadarzającej się okazji (tzn. w wielu miejscach naszego programu).

ZWYKŁE SKOKI - czyli ten sam pomysł, co powyżej, ale zrealizowany za pomocą zwykłego rozkazu JMP adres. Zasada jest prosta - JMP skacze do następnego JMP, ten z kolei do następnego, itd. ... co jakiś 50-ty skok należy wykonać jakąś operację ważną z punktu widzenia działania programu, ...i skakać sobie dalej. Gwarantowane, że tak z 200, 300 razy po całej pamięci i hackerowi będzie się robiło niedobrze na sam widok instrukcji JMP!

NEGACJA BAJTÓW - czyli po prostu procedura, która wykonuje instrukcję EOR (ExclusiveOr) dla zadanego obszaru pamięci a najlepiej dla obszaru, który tak zdekodowany stanie się dalszym ciągiem wykonywanego aktualnie programu. Jak zwykle potęga blokady zależy od ilości powtórzeń tego triku w naszym programie.

JMP (WEKTOR) - wszyscy znają działanie tego rozkazu, prawda? Jeśli tak, to znajdą tu na pewno pole do popisu. Jest jednak pewien kwiatek na tej łączce i to grubego kalibru (jego instalacja jako zabezpieczenia wymaga sporo wiedzy fachowej i doświadczenia, ale dobrze jest wiedzieć, że coś takiego istnieje, nawet jeśli nie będziecie w stanie tego wykorzystać). Otóż wyobraźcie sobie, że śledzicie właśnie pracę jakiegoś programu i nagle napotykacie na np. taki rozkaz:

```
JMP ($DC04)
```

I co teraz??! Komórki o adresach $DC04 i $DC05 zawierają zegar (o dość szybko zmieniających się wartościach) i z punktu widzenia „grzebiącego" w programie hackera, mogą tam się ukazać w zasadzie wartości zbliżone do losowych...! Autor zabezpieczenia oczywiście wiedział, pod jaki adres uda się potem procesor. Na pocieszenie powiem, że istnieje pewna bardzo prosta metoda na wykrycie, gdzie skoczył procesor....

A teraz parę słów o drugiej grupie, którą oznaczyliśmy literą B. W odróżnieniu od przedstawicieli grupy poprzedniej, blokady te mają za zadanie nie tyle utrudnienie uruchomienia, co wręcz zablokowanie startu

w przypadku stwierdzenia jakichś zmian w systemie, które nie zostały przyjęte jako „bezpieczne" dla pracy programu. Są to wszelkiej maści:

BLOKADY TESTUJĄCE DYSK, z którego został wgrany program. Najczęściej sprawdzają one, czy w wybranym miejscu na dysku jest coś zapisane w odpowiedni (najczęściej niestandardowy) sposób. Zmiany takie na ogół umykają uwadze osobie, która naiwnie sądzi, że kopia jest jak oryginał. „Połamanie" takich zabezpieczeń wymaga jednak sporej wiedzy, bo nie zawsze dobry program kopiujący typu „nibbler" jest w stanie z tym sobie poradzić.

TESTY NA OBECNOŚĆ CARDRIDGE'A - są już teraz prawie na porządku dziennym. O ile w przypadku prostych przystawek typu FINAL II czy FINAL III wykrycie nie przedstawia większego problemu (wystarczy „zerknąć" pod adres $DE00 do $DFFF aby się o tym przekonać - „to" tam zawsze jest, nawet po rozkazie „kill"!!!), to w momencie użycia czegoś lepszego, jak np. ACTION REPLAY, sprawa staje się o wiele bardziej skomplikowana.

Najczęściej rozwiązuje się ten problem pisząc procedurę, która wywoływana odpowiednio często z poziomu programu zabezpieczanego bedzie testowała, czy jest to wersja oryginalna, czy też „freezowana". W praktyce testuje się zużycie stosu (procedura „zamrażania" programu jest bardzo „stoso-żarłoczna") i w przypadku stwierdzenia dużych „ubytków" najczęściej kasuje całą pamięć. Może jeszcze jedna uwaga: aby uniemożliwić uruchomienie programu z aktywnym jakimkolwiek cardridge'm, wystarczy stwierdzić, że „coś" jest podłączone do Expansion Port. Aby lepiej zrozumieć zasadę działania tej blokady trzeba wiedzieć, że w zasadzie wszystkie (znane mi) karty umożliwiające „zamrażanie" programów korzystają z pewnego obszaru pamięci, który umożliwia wygodne przełączanie banków pamięci w danej karcie. Obszar ten leży pomiędzy adresami $DE00 i $DFFF. Jeśli podłączony jest jakiś cardridge, to w tym obszarze (przy wartości $37 w komórce $01) zawarte dane są stałe i na ogół są łatwo rozpoznawalne jako program w assemblerze. W przeciwnym wypadku każda próba odczytu da nam inną wartość. Można to wykorzystać do napisania prostej blokady:

```
        LDA $DE10      ;pobieramy wartość jednej
                       ;z komórek
        LDX #$07
        LDY #$00
LOOP1   CMP $DE10      ;i porównujemy to z kolejnym
                       ;odczytem
        BNE LOOP3      ;gdy wartość się różni skocz
                       ;do  LOOP3
LOOP2   DEY
        BNE LOOP2      ;krótka pętla czasowa
        DEX
        BNE LOOP1      ;licznik ilości sprawdzeń (tu: 7)
        JMP RESET      ;„coś" jest - kasuj komputer
LOOP3   JMP DALEJ      ;wartości są zmienne - nic nie jest
                       ;podłączone  -  można
                       ;kontynuować    program
```

TESTY KONTROLNE PAMIĘCI pozwalają z kolei na wychwycenie nawet drobnych modyfikacji, braku „resztek" z oryginalnego loadera, inną zawartość ekranu w momencie uruchamiania, inną pozycja kursora (tzn. inna niż po grzecznym wpisaniu LOAD„*",8,1),itp. I tradycyjnie, po stwierdzeniu zmian następuje kasowanie pamięci i pa,pa, „połamany" programiku.

Jak widać możliwości jest całe mnóstwo i mam nadzieję, że to zachęci Was do wymyślania i pisania własnych zabezpieczeń? A może chcielibyście się podzielić Waszymi własnymi osiągnięciami w tej dziedzinie? Czekamy na listy!

Paweł Sołtysiński

UWAGA UŻYTKOWNICY PROGRAMU VOICETRACKER V 4.0!

Niektórzy z Was sygnalizują, że nie wiedzą jak uruchomić demonstracje muzyczne znajdujące się na dyskietce lub taśmie. Otóż demonstracje te należy wgrywać nie jako samodzielne programy, ale jako moduły programu. Po wgraniu Voicetracker'a i wejściu do głownego okna edycji wybieramy opcję taśma lub dysk, a następnie wprowadzamy komendę LOAD (klawisz „l"). Komputer zapyta o tytuł pliku jaki chcemy wprowadzić (wpisujemy np. TUNE 01), a następnie wczyta go do pamięci. Po przejściu do głównego menu klawiszem F1 uruchamiamy daną demonstrację.

REDAKCJA

W związku z pojawiającymi się kłopotami w dystrybucji oferowanych przez nas dyskietek i taśm (wynikającymi z nieczytelnego bądź niekompletnego wypełnienia blankietów wpłat) przedstawiamy obok specjalny druk. Blankiet ten może służyć jako zamówienie i dowód wpłaty dla wszystkich oferowanych przez nas usług: sprzdaż dyskietek i taśm PDP, Voicetracker'a, zamówienie ogłoszeń itd.

REDAKCJA

UWAGA!

W związku ze wzrostem opłat pocztowych
(nasze pismo jest już grubsze i cięższe,
a więc musimy za jego wysyłkę płacić więcej)

z przykrością informujemy,
że od 1 sierpnia cena 1 egzemplarza „64 plus 4"
w prenumeracie wynosi 5000zł.

Przypominamy, że prenumeratę można zawrzeć na dowolny okres (nie krótszy niż dwa miesiące) do końca bieżącego roku. Na CZYTELNIE wypełnionych blankietach wpłat prosimy dopisać „PRENUMERATA" oraz okres jej trwania. Wpłaty prosimy przesyłać wykorzystując zamieszczony obok blankiet.
Przepraszamy.

Odcinek dla wpłacającego
Zł ....
słownie ....
wpłacający ....
....
....
(dokładny i CZYTELNY adres)
na rachunek:
Przedsiębiorstwa ABUK sp. z o.o.
87-100 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.
Opłata
zł.........

Odcinek dla Banku
Zł ....
słownie ....
wpłacający ....
....
....
(dokładny i CZYTELNY adres)
na rachunek:
Przedsiębiorstwa ABUK sp. z o.o.
87-100 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.
Opłata
zł.........

Odcinek dla posiadacza rachunku
Zł ....
słownie ....
wpłacający ....
....
....
(dokładny i CZYTELNY adres)
na rachunek:
Przedsiębiorstwa ABUK sp. z o.o.
87-100 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.
Opłata
zł.........

Odcinek dla Poczty
Zł ....
słownie ....
wpłacający ....
....
....
(dokładny i CZYTELNY adres)
na rachunek:
Przedsiębiorstwa ABUK sp. z o.o.
87-100 Wąbrzeźno, ul. 1 Maja 33,
Bank PKO SA Bydgoszcz, konto:
5.09011-400522.7-136-11-111.0.
Opłata
zł.........

TREŚĆ ZAMÓWIENIA:

....

TREŚĆ ZAMÓWIENIA:

....

Prosimy o CZYTELNE wypełnienie.

TREŚĆ ZAMÓWIENIA:

....

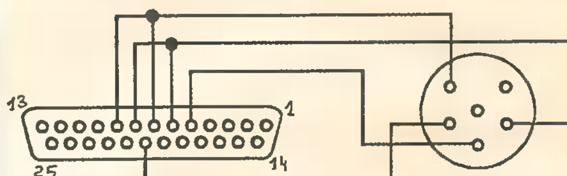
Prosimy o CZYTELNE wypełnienie.

PAMIĘTAJCIE!

Czytelne i dokładne wypełnienie wszystkich rubryk gwarantuje szybką i poprawną realizację Waszego zamówienia!

DZIĘKUJEMY!

UWAGA! W czerwcowym numerze naszego pisma, w artykule AMIGA + C-64 mylnie wydrukowano schemat przewodu połączeniowego. Widok przedstawiał gniazda od frontu, a nie jak podano od strony punktów lutowniczych. Przepraszamy!



SŁOWNIK SLANGU

Cześć! Ktoś może mi zarzucić, że nie lubię literki 'C' bo pomijam hasła zaczynające się właśnie na nią. Otóż chcę w tym miejscu publicznie odrzucić te wstrętne pomówienia. Ja kocham literkę 'C' lecz... nie pamiętam żadnych haseł !?!?! Dlatego właśnie dziś zaprezentuję Wam literkę 'D'!

Dana - Oj Dana, Dana wszystko przez Reagan'a!

Data - to, że Anglicy są trochę tego... wiedzą wszyscy, ale żeby na dane mówić Data, na Datę - Date, na manetkę - joystick a na sprzęg - interface to już jest szczyt wszystkiego.

Date - jeżeli masz zegarek z datownikiem to zawsze możesz liczyć na w miarę aktualną Datę, jeżeli nie to nigdy nie sugeruj się tym co podaje ci komputer.

Delete - po wykonaniu tej funkcji stwierdzisz, że właśnie zniknął ci gdzieś najbardziej potrzebny program (ale to chyba nie wynika z zasady działania tej funkcji...???).

Delira - proces degeneracji umysłu zachodzący najczęściej w czasie trwania różnorodnych Copy-Party i dający swój wyraz w idiotycznych tekstach, jedzeniu dyskietek, kodowaniu nogą, wkładaniu dwóch dyskietek do stacji, rzucaniu się hamburgerami, gryzieniu joystick'ów i rąk kolegów, piciu octu zamiast Coca-Coli, itp...

Denise - wstrętny i niebezpieczny układ kontrolujący wizję w Amidze; w nowych Amigach (np. 3000) układ ten potrafi uszkodzić monitor!!!

Density - termin oznaczający gęstość zapisu danych na np. dysku magnetycznym. Dyski pojedyńczej gęstości w zasadzie wyszły już z obiegu. Pozostały tylko dyski podwójnej (2D) i wysokiej (HD) gęstości.

Demoa - jedne z najpopularniejszych programów na komputery firmy Commodore w których programiści i koderzy prezentują swoje umiejętności często zawstydzając profesjonalistów i przyprawiając o ból głowy konstruktorów sprzętu, którzy z wyciem „to nie ma prawa działać!" wyskakują z okien.

Digital - wszystko co można zapisać bitami jest digital (wszystko można zapisać bitami!)

Digitalizer - zapisze nam WSZYSTKO! bitami do pamięci - a potem rób z tym co dusza zapragnie (możesz nawet skasować).

Disassembler - z normalnych cyferek robi ciąg jakiś idiotycznych słów (dla C=64 np. STA, LDA..., a dla Amigi np. move, lea), które i tak nie mają nic wspólnego z normalnym językiem (BASIC?) a na dodatek występują w kombinacjach z różnymi jeszcze bardziej idiotycznymi cyferkami - zrozum tu człowieku te głupie maszyny!

Disk - niezwykle pakowny kawałek folii o wymiarach od 3 do 8 cali wsadzony w plastikową okładkę, mieszczący około tony produktów chorej wyobraźni.

Do d... - termin, którym określa się jakość programów napisanych przez konkurencyjną grupę.

Dos - element oprogramowania (siedzący najczęściej w Rom'ie), który umożliwia nam w większości przypadków komunikację z dyskiem (pochodzi od angielskiego Disk Operating System) i w skrajnych przypadkach tj. gdy posiadamy jedyną wersje naszego programu lubi poczyniać małe Read-Write Error'y właśnie na tym zbiorze. DOS'y dzielimy na działające jako tako (Amiga Dos) i niedziałające w ogóle (patrz - TOS).

Drink - element przyspieszający wejście w stan deliry (patrz - Delira).

Drive - tu wsadź dyskietkę (patrz - Disk)! W przypadku kłopotów z wyjmowaniem radzimy używać odpowiednio wygiętego widelca!

Drum - sample gara (patrz - Gar).

Duddie - tego się nie da opisać, to trzeba samemu zobaczyć.

Duszki - w gwarze aTAROWIANÓW tak okre[illegible] normalne sprite'y - i kto mi powie, że oni są normalni.

Dynamic Hi-Res - nowy tryb graficzny Amigi (emulowany programowo), od którego póły spadają (patrz Póły).

Dysk - to co Disk tylko w wersji polskiej - [illegible] udane spolszczenie - może być w wersji tw[illegible] lub miękkiej.

Dyskietka - jeszcze ładniejsza nazwa dla giętkiego płaskokrążka informacjonośnego (za Komputer'em) ważąca po wypełnieniu od 360 kg do 2 ton.

Dziewczyna - miejsce odpoczynku po skończonej pracy z komputerem.

Suplement:

Gar - często używany instrument muzyczny - nadaje rytm wszystkiemu co gra obok niego - przez niektórych muzyków wykorzystywany jako jedyny instrument w muzyczkach.

Póła - hackerskie określenie szczęki - naturalnej lub sztucznej, która w kontakcie audiowizualnym z komputerami często lubi zmieniać swoją pozycję pionową przy czym po osiągnięciu minimum wydaje dźwięk słyszany w połowie małego miasteczka.

TOS - Trash Operating System instalowany w aTARI ST (Super Trash)

Tym razem słownik slangu przygotował

Marcin Dudar

(to chyba widać nie?) i czasami coś dopisał

Sambor Kuźma.

Cześć!

Kącik początkującego kodera (cz. 4)

W dzisiejszym odcinku nauczymy się stawiać „Bary", i poznamy kolejne rozkazy procesora MOTOROLA 68000.

SUB Dn,<adres efektywny > „Subtract Binary" - odejmowanie
<adres efektywny>, Dn

Instrukcja ta odejmuje operand źródłowy od operandu przeznaczenia i tam też umieszcza wynik. Możemy wykonywać operację na bajcie, słowie i długim słowie.

Znaczniki: X - ustawiany, gdy wystapiła pożyczka
N - ustawiany, gdy najstarszy bit operandu jest ustawiony
Z - ustawiany, gdy wynikiem jest zero
V - ustawiany, gdy wystapi nadmiar
C - analogicznie jak X

W przeciwnych przypadkach do wyżej opisanych znaczniki zostaną skasowane.

Przykład:

sub.l d0,(a0)+ - od zawartości komórki wskazywanej przez rejestr adresowy a0 zostanie odjęta zawartość rejestru d0. Wynik umieszczony będzie w tej samej komórce, a rejestr a0 zwiększony o 4.

SUBA <adres efektywny>, An „Subtract Address" - odejmowanie adresu.

Za pomocą tej instrukcji możemy od adresu umieszczonego w rejestrze adresowym odjąć wartość określoną adresem efektywnym. Wynik operacji zostanie umieszczony w rejestrze adresowym. Rozmiarem tej operacji może być słowo lub długie słowo.

Znaczniki: wszystkie pozostaną nie zmienione.

Przykład:

suba.l #$10000,a6 - od rejestru a6 zostanie odjęta wartość $10000

SUBI #liczba,<adres efektywny> „Subtract Immediate" - odejmowanie natychmiastowe.

Instrukcja ta odejmuje daną natychmiastową od adresu efektywnego i tam też umieszcza wynik. Rozmiarem operacji może być zarówno bajt, słowo jak i długie słowo.

Znaczniki: analogicznie jak SUB

Przykład:

subi.b #$20,d0 - wartość $20 zostanie odjęta od bajtu rejestru d0. Wynik zostanie umieszczony w d0.

SUBQ #liczba,<adres efektywny> „Subtract Quick" - odejmowanie szybkie.

Instrukcja ta odejmuje natychmiastową wartość z zakresu od 1 do 8 od adresu efektywnego. Rozmiarem tej operacji może być bajt, słowo lub długie słowo.

Znaczniki: analogicznie jak SUB.

Przykład: subq.l #3,d0 - wartość 3 zostanie odjęta od rejestru d0

SUBX Dx,Dy „Subtract with Extend" - odejmowanie z rozszerzeniem
-(Ax),-(Ay)

Instrukcja ta odejmuje operand źródłowy od operandu przeznaczenia z uwzględnieniem znacznika X. Rozmiarem operacji może być bajt, słowo lub długie słowo. Możliwe do użycia są tylko dwa tryby adresowania: predekrement i rejestru danych.

Znaczniki: analogicznie jak SUB

Przykład:

sub.l d0,d1 - od rejestru d1 zostanie odjęty rejestr d1. Wynik zostanie umieszczony w rejestrze d1.

ROR #liczba,Dx „Rotate Right" - obrót w prawo
Dx,Dy
<adres efektywny>

Instrukcja ta jest analogiczna do instrukcji ROL (patrz poprzedni odcinek), z tą jednak różnicą, że obrót wykonywany jest w prawo.

```
——→|b7|b6|b5|b4|b3|b2|b1|b0|————→ C
— — — — — — — — — ← — — — — — — — — —
```

Tryby adresowania i rozmiar operacji - tak samo jak ROL.

Znaczniki: X - nie zmieniany
N - modyfikowany zgodnie z najstarszym bitem operandu
Z - ustawiany, gdy wynikiem obrotu jest zero
V - zawsze zerowany
C - zgodny z ostatnio wysuniętym bitem

Przykład:

ror.l #2,d7 - rejestr d7 zostanie obrócony o 2 bity
ror.w d2,d3 - rejestr d3 zostanie obrócony o tyle bitów, jaką zawiera wartość d2
ror.b d4 - rejestr d4 zostanie obrócony o 1 bit

BCHG Dx,<adres efektywny> „Test A Bit And Change" - testuj bit i zamień.
#liczba,<adres efektywny>

Instrukcja ta testuje określony bit operandu, odpowiednio modyfikując znacznik Z. Następnie testowany bit zostaje zanegowany.

Instrukcja ta może mieć dwa rozmiary:

1. Długie słowo - wtedy numer bitu (od 0 do 31), który ma zostać przetestowany podajemy w źródłowym rejestrze danych.

2. Bajt - numer bitu w zakresie od 0 do 7 podajemy jako liczbę natychmiastową.

Znaczniki: X,N - nie zmieniane
Z - ustawiany, gdy testowany bit był skasowany
V,C - nie zmieniane

Przykład:

bchg #3,d0 - zostanie przetestowany 3 bit rejestru d0.
move.l #$10,d3
bchg d3,$10000 - w rezultacie działania tych dwóch rozkazów zostanie przetestowany $10 bit długiego słowa umieszczonego pod adresem $10000.

BCLR Dx,<adres efektywny> „Test A Bit And Clear" - testuj bit i skasuj
#liczba,<adres efektywny>

Instrukcja ta testuje określony bit, a następnie kasuje go. Rozmiar i tryb adresowania tak samo jak BCHG.

Znaczniki: analogicznie jak BCHG

Przykład:

bclr #6,d0 - zostanie przetestowany, a następnie skasowany 6 bit rejestru d0.

BSET Dx,<adres efektywny> „Test A Bit And Set" - testuj bit i ustaw

#liczba,<adres efektywny>

Za pomocą tej instrukcji możemy przetestować określony bit, a następnie ustawić go. Rozmiar i tryb adresowania tak samo jak BCHG.

Znaczniki: analogicznie jak BCHG

Przykład:

```
bset #0,(a0) - zostanie przetestowany, a następnie
ustawiony zerowyy bit bajtu w pamięci znajdujący się
pod adresem wskazywanym przez rejestr a0.
```

BTST Dx,<adres efektywny> „Test A Bit" - testuj bit

#liczba,<adres efektywny>

Instrukcja ta testuje określony bit operandu nie zmieniajac go.

Rozmiar i tryb adresowania analogicznie jak BCHG.

Znaczniki: tak samo jak BCHG.

Przykład:

```
btst #6,$bfe001 - zostanie przetestowany szósty bit rejes-
tru $bfe001. Bit ten ma za zadanie kontrolować lewy przy-
cisk myszy. Jest on ustawiony, gdy nie przyciskamy,
natomiast zostaje skasowany w przypadku
naciśnięcia tego przycisku.
```

I jak zwykle na koniec program. W wielu programach demonstracyjnych zauważyć można stojące lub poruszające się rurki pospolicie zwane barami (ang: bar - drążek). Są one stosunkowo łatwe do wykonania nawet dla początkującego kodera.

Całe zadanie sprowadza się do wykonania odpowiedniego Copper List'u (patrz: Zmagania z Copper'em - w jednym z poprzednich numerów). Wymaga to jednak wpisania wielu liczb, ponieważ w niektórych demach długość takich Copper List'ow sięga nawet kilkunastu kilobajtów danych. Aby uniknąć tej pracochłonnej roboty napiszemy procedurę, która zrobi wszystko za nas.

Aby zmienić kolor w linii ekranu potrzebujemy dwie instrukcje Copper List'u - instrukcji WAIT, która odczeka na określoną pozycję ekranu, oraz instrukcje MOVE, która w danej pozycji zmieni kolor. Dla naszych barów instrukcje te będą miały nastepujący format:

```
dc.l $yy07fffe     ;Wait - czekaj na pozycje yy
dc.l $01800xxx     ;Move - prześlij kolor o kodzie xxx
```

Po każdej instrukcji WAIT następować będzie instrukcja MOVE, a po tej ostatniej następna instrukcja WAIT - czyli oczekiwanie na nastepną pozycje (yy zostanie powiekszone o 1) w celu ponownej zmiany koloru instrukcj MOVE itd...

```
        lea      BarArea(pc),a0      ;inicjacja - nadanie
                                     wartości początkowych
        lea      Colours(pc),a1
        move.l   #$3007fffe,d0       ;zaczynamy od linii $30

Loop:                                ;pętla główna tworzenia
                                     Copper List'u
        move.l   d0,(a0)+            ;instrukcja WAIT do
                                     bufora
        move.w   #$0180,(a0)+        ;starsze słowo instru
                                     kcji MOVE do bufora
        move.w   (a1)+,(a0)+     ;pobieranie koloru z tabeli
                 i przesyłanie do bufora - młodsze słowo MOVE
        addi.l   #$01000000,d0   ;przekształcamy instrukcje
                 WAIT aby wskazywała na następną linie
        cmpi.l   #$6007fffe,d0   ;czy linia $60
        bne      Loop            ;jeśli nie, to skok do Loop -
                                 twórz nastepną
        move.l   #Copper,$dff080  ;ustawiamy gotowy
                                  Copper List
Wait:
        btst     #6,$bfe001      ;powrót po przyciśnięciu
                                 lewego przycisku myszy
        bne      Wait
        rts

Copper:
        dc.l     $00960020       ;ustawienie DMA
        dc.l     $01080000,$010a0000    ;modulacja ekranu
        dc.l     $01000000, $01020000, $01040000
                 ;zerujemy rejestry kontroli
        dc.l     $008e2c50,$00902cc1,$00920038,$009400d0
                 ;standardowa wielkość ekranu BarArea:
        blk.l    96,0       ;Obszar na linie zmiany kolorów
        dc.l     $fffffffe  ;Koniec Copper List'u
Colours:
  dc.w $0000,$0002,$0004,$0006,$0008,$000a,$000c,$000e
                 ;kody kolorów dla barów
  dc.w $000e,$000c,$000a,$0008,$0006,$0004,$0002,$0000
  dc.w $0000,$0022,$0044,$0066,$0088,$00aa,$00cc,$00ee
  dc.w $00ee,$00cc,$00aa,$0088,$0066,$0044,$0022,$0000
  dc.w $0000,$0202,$0404,$0606,$0808,$0a0a,$0c0c,$0e0e
  dc.w $0e0e,$0c0c,$0a0a,$0808,$0606,$0404,$0202,$0000
```

Powyższy program stworzy linie do zmiany kolorów o numerach z przedziału od $30 do $60. łatwo można jednak zmodyfikować to wpisując własne numery linii w odpowiednie miejsca. Aby uzyskać inne kolory należy zmienić wartości danych za etykietą 'Colours'.

Krzysztof „K.K." Kobus

Zapraszamy wszystkich do udziału
w stałym konkursie pod hasłem:

Najlepszy program miesiąca

W konkursie udział mogą brać wszyscy, którzy nadeślą własne, nigdzie nie publikowane prace.

Tematyka programów dowolna.

Konkurs rozgrywany jest osobno dla komputerów **C-64 i AMIGA.**

Teksty programów należy nadsyłać na adres redakcji na dyskietce lub w postaci czytelnego rękopisu (dyskietki będą przez redakcję zwracane).

Objętość programu wraz z opisem i komentarzem nie powinna przekraczać 4 stron maszynopisu.

Raz w miesiącu Sąd Konkursowy wybierze najlepsze programy przyznając ich autorom dwie główne nagrody po **500.000** zł każda. Decyzje Sądu Konkursowego są nieodwołalne. Oprócz zdobycia głównej nagrody autorzy mają szansę na publikację swych prac na łamach naszego pisma.

Pracę prosimy podpisywać imieniem i nazwiskiem oraz dokładnym adresem autora.

Redakcja

TRACK WRITE

Na piątym dysku „64+4 Public Domain" znajduje się program Track Write v1.02 autorstwa Carnivore. Program ten służy do instalacji własnych programów ściąganych po zabootowaniu dysku (np. intro lub wybieraczka do programów).

Po wczytaniu programu pojawia się okienko z kilkoma gadżetami:

Filename.

Po kliknięciu w okienku przy Filename, możemy wpisać tam nazwę programu, który chcemy wczytać i po wciśnięciu klawisza Return program ten zostanie załadowany.

Destination.

Oznacza dysk, na którym zostanie nagrane nasze intro - standardowo ustawione jest DF0, ale klikając w okienku możemy wpisać tam DF1 lub cokolwiek innego.

Start Block.

Jest to blok na dysku, od którego nasze intro zostanie zapisane (minimalna wartość to 2 - przecież nie możemy zapisać go na bootblock'u).

Last Block.

To ostatni blok dla którego zostanie zajęte miejsce na dysku.

Filesize.

Rozmiar wczytanego programu i ewentualnie jaka ilość pamięci ma być później rezerwowana.

Load address.

Adres pod który zostanie załadowane intro.

Start address.

Adres do którego loader wykona skok po wczytaniu programu (najczęściej adres ładowania zbioru).

WRITE BOOTBLOCK.

Jeżeli ta opcja jest aktywna to dodatkowo oprócz intra zostanie nagrany bootblock, który załaduje to intro i wykona skok do niego.

ALLOCATE MEMORY.

Jeżeli nagrywamy bootblock to może on także zarezerwować obszar pamięci, w który wczytane będzie intro (aby nic na nie nie „weszło").

ALLOCATE BLOCKS.

Zarezerwuje te bloki na dysku, na których znajduje się intro aby przypadkowo nie nagrać na nie jakiegoś zbioru.

WRITE TRACKS TO DISK.

Po kliknięciu na ten gadżet, program rozpocznie pracę, tj. nagra program na dysk, ewentualnie nagra bootblock i zaallokuje bloki.

Życzę miłej zabawy z tym programem i oby nie sprawiał nikomu kłopotów (a oczywiście źle obsługiwany może je sprawić...)

Marcin „Duddie" Dudar

Własne demo - Amiga

Cykl artykułów „Jak zrobić własne demo" na Commodore 64 cieszy się dużym powodzeniem, dlatego też postanowiliśmy stworzyć podobny na AMIGĘ. Piszcie do redakcji, co o tym sądzicie, czy ta tematyka jest potrzebna. Dostosujemy się do Waszych wymagań. Zaczniemy może trochę od końca, gdyż dzisiaj napiszę ja wyjść z dema do systemu.

Wczytując demo i uruchamiając je ingerujemy w pracę systemu. Zmieniamy wartość różnych komórek umieszczając w nich potrzebne nam wartości. Aby komputer mógł poprawnie pracować po wyjściu z programu musimy odtworzyć pierwotne wartości rejestrów. Chodzi tu głównie o rejestry odpowiadające za stan przerwań, rejestry COP1LCH i COP1LCL - czyli adres „copper list", oraz rejesrty audio.

1. Rejestry przerwań. Koderzy w swoich demach z reguły nie używają przerwań, gdyż zabierają one zbyt dużo czasu ekranowego. Dlatego też zaraz na samym początku zabraniają ich wykonywanie czyli mówiąc slangowo „zabijają je". Wpierw jednak należy odczytać ich stan aby przy wychodzeniu odtworzyć ich pierwotną zawartość. Robimy to w następujący sposób:

```
lea       $dff000,a6
more.w    $1c(a6),Mem
more.w    $02(a6),Mem+2
or.l    # $80008000,Mem
more.w  # $7fff,$9a(a6)
```

Te pięć instrukcji przygotowuje przerwania do późniejszego odtworzenia i następnie blokuje je. Tę sekwencję należy umieścić zaraz na początku dema. Natomiast przy wychodzeniu należy wykonać:

```
lea       $Dff000a6
moer.w    Mem,$9a(a6)
more.w    Mem+2,$96(a6)
```

oraz umieścić:

```
Mem: dc.l 0
```

Czynności opisane w punktach 2, 3 i 4 wykonujemy przy wychodzeniu z dema.

2. Adres aktualnie używanego przez system „Copperlist" znajduje się pod adresem $26 od bazy „Graphics.library". Nalćży więc otworzyć tę bibliotekę i skopiować długie słowo spod adresu będącego sumą wskaźnika bazy i $26 do rejestru $Dff080.

3. Następną czynnością, którą należy wykonać jest wyciszenie muzyki. W tym celu skaczemy do programu „mt_end" w ogólnie dostępnym grajku do Soudrachera /Noise-trackera. W przypadku korzystania z innego, należy poszukać analogicznego podprogramu.

4. Aby ostatecznie powrócić do systemu (po wykonaniu w/w czynności) czyścimy rejesrt d0 za pomocą „clr.l d0" - tym samym informując system, że wszystko jest OK i wykonujemy zwykłe „rts".

Należy również pamiętać, aby w pętli głównej naszego programu sprawdzać stan np. lewego przycisku myszy (btst #6,$bfe001) i w zależności od niego skoczyć do procedury powrotu.

Krzysztof „K.K."Kobus

POLICE QUEST I
czyli zabawa w policjantów i przestępców

Po raz kolejny witam wszystkich miłośników gier przygodowych i zapraszam na może trochę starą, ale ciekawą wyprawę. Dziś wcielimy się w postać nieustraszonego Sony Bonda, który jest policjantem w miasteczku Lytton.

Grę rozpoczynamy w głównym holu posterunku policji. Zanim cokolwiek zrobisz rozejrzyj się trochę i poznaj okolicę. Teraz do dzieła. Idź do szatni i otwórz swoją skrytkę. Zabierz z niej wszystkie przedmioty potrzebne do twojej pracy (teczkę też). Teraz jak co rano udaj się do sali odpraw, by otrzymać rozkazy. Nie zawadzi zaglądnąć do skrytki i przeczytać wiadomość. Ponieważ przyszedłeś trochę wcześniej możesz spokojnie poczytać gazetę. Kiedy skończysz czytać znajdź swoje miejsce i w skupieniu wysłuchaj całej odprawy. Po wyjściu z pokoju odpraw zaopatrz się w radiotelefon i kluczyki do twojego służbowego samochodu. Teraz na parkingu sprawdzasz czy ktoś nie spuścił ci powietrza z opon i już możesz się udać na patrol. Ponieważ nie masz nic lepszego do roboty proponuję abyś zwiedził miasto i sporządził sobie mapę z nazwami ulic i wazniejszych miejsc. I oto dostajesz pierwsze wezwanie w dniu dzisiejszym.

Na rogu Czwartej ulicy i Fig zdarzył się wypadek. Włączasz syrenę i już jesteś na miejscu. Wysiądź z samochodu i podejdź do wraku auta. Na pierwszy rzut oka to normalny wypadek, ale po głębszych oględzinach zastanawia cię dziura w szybie i w głowie ofiary. To może być morderstwo! Natychmiast powiadamiasz o tym centralę, która wysyła brygadę z wydziału zabójstw. W tym czasie porozmawiaj z gapiami. Od młodego człowieka dowiesz się paru interesujących rzeczy (informuj o tym centralę). Po przybyciu detektywów możesz z powrotem ruszyć na swój patrol. Kolejne wezwanie to przerwa na kawę w barze Carol's. Tutaj zosteniesz powiadomiony co do danych dotyczących martwego kierowcy.

Teraz możesz ponownie udać się na ulicę. Po krótkiej jeździe dostrzegasz sportowy samochód, który przejechał na czerwonym świetle. Włącz syrenę i zatrzymaj go. Okazuje się, że kierowca to piękna dziewczyna. Jeśli nie chcesz stracić paru punktów to lepiej wypisz jej mandat, ale możesz się również z nią umówić (lepiej do niej później nie dzwoń!). Kolejne wezwanie do Wino Willy's. Z samochodu weź pałkę i idź porozmawiać z motocyklistami. W razie kłopotów możesz ich postraszyć.

Kolejna niespodzianka to samochód z zawianym kierowcą. Zatrzymaj go i daj do nadmuchania „balonik". Teraz zaobrączkowanego kierowcę odwieź do więzienia (pistolet zostaw w skrytce). W więzieniu dowiadujesz się, że zwolniło się miejsce w oddziale do spraw narkotyków i możesz złożyć swoje podanie. Wracasz do biura i po napisaniu podania udajesz się do Doodley'a. Nie przejmuj się incydentem z kurą. Jeśli masz ochotę możesz pojechać na imprezę do Blue Room (cywilnym samochodem i w cywilnych ciuchach). Zaraz jednak wracasz na odprawę w biurze. Sprawdź wiadomość w skrytce.

Po wyruszeniu na patrol otrzymujesz wiadomość, że w twoim kwadracie widziano poszukiwany samochód. W momencie gdy go zidentyfikujesz użyj syren by go zatrzymać. Teraz nie zgrywaj bohatera tylko wezwij posiłki. Nie daj facetowi powodów do strzału i po aresztowaniu przeszukaj jego i samochód, potem do więzienia z bandziorem.

W biurze dowiadujesz się, że przeniesiono cię do narkotyków. Przebierz się w cywilne ubranie i udaj się do nowego szefa. Kolejne wezwanie zaprowadzi cię do parku, gdzie odbędzie się transakcja. Schowaj się w krzakach i w odpowiednim momencie użyj radia i pistoletu. Zatrzymanych zawieź do więzienia.

Po powrocie do biura poczekaj na Laurę, która powie ci o Marii (ta słodka dziewczyna z baru). Udaj się ponownie do więzienia i spróbuj namówić Marię, by pomogła wam w operacji. W drodze powrotnej poproszą cie o zidentyfikowanie zwłok w Cotton Cove. Udaj się do szefa, który wprowadzi cię w temat operacji „Hotel". Następnie idź do łazienki i przefarbuj włosy, oraz zabierz nowe ubranie i wróć do szefa. Tutaj otrzymasz dalsze instrukcje oraz $1000 (nie zapomnij o numerze telefonu). Cadillac nadaje się w sam raz do twojego nowego wyglądu, więc nim udajesz się do hotelu. Tu wynajmij pokój i udaj się do baru.

Razem z Marią spróbuj nakłonić barmana by zaprowadził was do tajnego kasyna (w razie czego daj mu łapówkę). Teraz udaj się do swojego pokoju i zadzwoń do szefa by przysłał posiłki. Następnie zadzwoń po taksówkę (0 to informacja) i odeślij Marię na posterunek. Raz jeszcze udaj się do barmana, który zaprowadzi cię do szulernii. Usiądź przy ostatnim stoliku i zaproponuj partię pokera. Jeśli będziesz miał szczęście to szybko wygrasz odpowiednią sumę pieniędzy by zaproponowano ci grę o trochę większe stawki. Przed tym jednak udaj się do swojego pokoju, by zabrać radio od jednegoo ze swoich kolegów. Raz jeszcze poproś barmana by zaprowadził cię do sali gier.

Teraz ponownie musisz wygrać odpowiednią sumę pieniędzy by otrzymać propozycję dobrego interesu. Udaj się za mężczyzną do jego pokoju, lecz zanim tam wejdziesz użyj radia. Teraz nie pozastaje ci nic innego, jak usiąść wygodnie i obejrzeć skończenie gry. W przyszłym miesiącu zajmiemy się drugą częścią tej gry czyli dobierzemy się do „Police Quest II".

Mr.Raf

Przegląd gier

Tytuł: WINGS OF DEATH
Firma: Thalion/Eclipse
Code: Marc Rosocha
Grafika: Erik Simon
Muzyka: Jochen Hippel

Po raz kolejny firma Thalion dostarcza porcję dobrej rozrywki. Tym razem proponuje nam kolejną grę typu „Shoot'em up". Nie będę się wgłębiał w historię typu „za siedmioma górami..." lub „zły czarnoksiężnik zamienił cię..." gdyż nie o to nam chodzi.

Na początek parę szczegółów technicznych. Gra mieści się na dwóch dyskach i nie wymaga rozszerzenia pamięci. Za pomocą klawisza „Ctrl" można zmieniać organizację ekranu do systemu PAL lub NTSC (nieocenieni „łamacze"). Jeśli chodzi o samą grę to pomysł jest stary jak pierwsze automaty do gier. Pole gry przesuwane jest pionowo, a twój statek posiada możliwość rozbudowy poprzez łapanie „POW BALLS" (zasobniki) wypadających z zestrzelonych statków. Na końcu każdej strefy musisz pokonać wielkiego przeciwnika by móc przesuwać się dalej. Jest to typowy scenariusz wielu gier. Tym co można uznać za dobrą stronę „Wings of Death" to napewno ładna grafika i wyraźne sample. Muszę w tym miejscu zmartwić wszystkich posiadaczy monochromatycznych monitorów, że o ile w kolorze można spokojnie odróżnić wszystkie obiekty, to na zwykłym „zieleńcu" jest prawie niemożliwością normalna gra (sam ledwo doszedłem do 6 strefy). Bardzo wyraźne są sample, które przy wybranej opcji „MUSIC+" informują Cię jakie właściwości ma zasobnik, który wziąłeś. Są to przeważnie różne rodzaje broni (smoczy płomień, promień energetyczny i podobne). W miarę jak zbierasz zasobniki tego samego rodzaju zwiększa się siła twojej broni. Inne zasobniki to: energia, dodatkowe życie, prędkość statku lub bomba, która powoduje ubytek energii i siły twojej broni. Gra jest bardzo podobna do „Dragon Spirit".

GRAFIKA:	*6*
MUZYKA/FX:	*5*
ANIMACJA:	*5*
POMYSŁ:	*3*
PRZYJEMNOŚĆ GRANIA:	*6*
OGÓLNIE:	*5*
OCENA:	*MAŁY PLUS*

Tytuł: BATTLE COMMAND
Firma: Ocean/Realtime Software Games

Firma „OCEAN" zdobywa sobie coraz większą popularność wśród fanów gier komputerowych po przez swoje coraz lepsze produkty. Tym razem proponuje nam symulację czołgu nowej generacji. Sam scenariusz jest prosty i wymaga od nas ukończenia 15 misji, po których następuje finałowe zadanie kończące całą grę. Do naszej dyspozycji oddany jest czołg mogący przenosić każdy rodzaj broni od zwykłych pocisków do wielkich bomb czasowych. Jednak nie od samego początku otrzymujemy dostęp do wszystkich śmiercionośnych zabawek. W miarę jak zaliczamy coraz więcej misji uzyskujemy możliwość rozbudowy arsenału. Oprócz broni możemy otrzymać dodatkowe wyposażenie w postaci noktowizora lub silnej lunety, co pozwala nam identyfikować cele z dużej odległości. Innowacją w tej grze jest bardzo udane połączenie elementów strategiczno-symulacyjnych z typowo zręcznościowymi. Cała grafika oparta jest na dość dobrze i szybko przeliczanej grafice wektorowej (na uwagę zasługuje scena zrzutu czołgu na teren wroga). Również oprawa dźwiękowa wykonana jest z dużym wyczuciem. W czasie gry dostępne są same f/x'y, natomiast przed jej rozpoczęciem możemy posłuchać kawałka dobrej muzyki. Jest to gra, której nie można ukończyć w jeden wieczór i do której jeszcze nie raz z miłą chęcią będzie można powrócić.

GRAFIKA:	*7*
MUZYKA/FX:	*8*
ANIMACJA:	*6*
POMYSŁ:	*5*
PRZYJEMNOŚĆ GRANIA:	*7*
OGÓLNIE:	*7*
OCENA:	*DUŻY PLUS*

Tytuł: MURDER
Firma: U.S.Gold/Kingsley Harrison
Projekt gry i wykonanie:
Grant Harrison i Jason Kingsley

Doczekaliśmy się kolejnego przeboju w grach detektywistycznych. Po wspaniałym „Killed Until Dead" mamy następną przygodę, która zapowiada się całkiem ciekawie.

W wielkim domu „Ghastley Manor" zostało popełnione morderstwo i ty, jako jeden z najlepszych detektywów, zostałeś poproszony o wykrycie sprawcy, motywu i narzędzia zbrodni. Nie jest to sprawa łatwa, ale dzięki opcjom ci dostępnym możliwa do wykonania. Tak jak każdy prywatny detektw masz możliwość zbierania odcisków palców i ich porównywania z uprzednio już zebranymi. Możesz również zbierać dowody rzeczowe i przeglądać swoje prywatne zapiski. Jedną z najważniejszych możliwości to konwersacja z napotkanymi osobami. Można zadawać im pytania na temat pojedynczych osób, jak również o powiązania między nimi. Można nawet zadać pytanie co dana osoba robiła ze wskazaną rzeczą w określonym miejscu. Jednak zauważyłem że mieszkańcy zamku nie są zbyt rozmowni i ciężko od nich wyciągnąć informacje.

Parę słów jeżeli chodzi o szczegóły techniczne. Grafika wykonana jest bardzo oszczędnie lecz spełnia swoje zadanie. Jedynie można jej zarzucić zbytnią monotonność. W muzyce w zasadzie mamy do czynienia tylko z f/x'ami. Są one bardzo pomysłowe np. szepty w momencie gdy przepytujesz jedną z rozmawiających osób.

Gra jest warta spędzenia nad nią kilku nieprzespanych nocy.

GRAFIKA:	*4*
MUZYKA/FX:	*6*
ANIMACJA:	*4*
POMYSŁ:	*5*
PRZYJEMNOŚĆ GRANIA:	*6*
OGÓLNIE:	*5*
OCENA:	*MAŁY PLUS*

Oceniał:
Mr. Raf

Gracz doskonały

DRAGON SPIRIT

Zatrzymać grę (F9), wpisać DRAGON HEAD i wystartować ponownie (F10)

DRAGON SCAPE

Nacisnąć TAB i „2" aby przeskoczyć do następnego poziomu.

ZANY GOLF

W grze jest tajna komnata. Na ostatnim poziomie jest mysia dziura, w której czasami pojawiają się oczy. Kiedy staną się one czerwone wrzuć piłeczkę do dziury. Znajdziesz się w tajnym poziomie „Mystery".

FUSION

Na tablicę wyników wpisz SWAMP THING, następnie naciśnij „E". Teraz możesz zmieniać poziomy klawiszami + i -.

ROAD BLASTERS

Na starcie wpisz LAVILLASTRANGIATO. Masz teraz do sypozycji następujące opcje:

S - następny poziom
P - tankowanie
G - koniec gry
1 - UZI
2 - pociski Cruise
3 - osłona
4 - dopalacze
0 - wyłączenie wszystkich broni.

OOOPS UP

Kod do ostatniego setnego poziomu: 4799

GOLDRUNNER

W tabeli wyników wpisz „easymode". Za pomocą klawisza F9 możesz teraz przeskakiwać poziomy.

LEMMINGS

Poniżej podajemy kilkanaście kodów dla różnych poziomów zdobytych w ciągu kilku nieprzespanych nocy. Pamiętajcie - wiele Lemingów straciło życie abyśmy mogli zdobyć te informacje!

FUN - HNNHCMOEDN
NNHCAKLFDS
BMNNNNLIDJ
FLCMJLMODR
TRICKY - NHLDMCEDEN
EMCMJLLGEJ
KJHMDMCKEX
KJKLGOCCFX
MCANLLGHFV
HMGMCMOMFV
TAXING - GMCMKLMOFW
ICCNNONPFX
GCKINLHBGS
OKNLLFCDGN
HLDNGKOEGM
GAJHMMHJGW

Miłej zabawy życzą

Hi-Man & Duddie

O BLITTERZE SŁÓW KILKA!

Amiga została wyposażona przez konstruktorów w potężne narzędzie jakim jest Blitter - procesor graficzny wykonujący operacje logiczne na pamięci.

Blitter może wykonywać takie operacje jak: AND (logiczne I), OR (logiczne LUB), negacja oraz przesuwanie o zadaną ilość bitów. Działa na zasadzie przenoszenia danych z jednego miejsca pamięci w inne jednocześnie dokonując zadanej operacji. Posiada on trzy komórki zawierające adresy źródeł (Sources) i jedną komrkę zawierającą adres przeznaczenia (Destination). Z trzech komórek źródeł możemy jednocześnie wysyłać dane pod jeden adres przeznaczenia. Portami źródłowymi są: A, B i C a portem przeznaczenia jest D.

Porty te mają adresy (wpisujemy długie sowo):

```
BLTAPT    - $dff050
BLTBPT    - $dff048   adresy źródeł ABC
BLTCPT    - $dff04c
BLTDPT    - $dff054   adres przeznaczenia D
```

Dla każdego z portów możemy ustawić inne modulo (patrz modulo w CopperList) i umieszczamy je w komórkach (wpisujemy sowo):

```
BLTAMOD   - $dff064
BLTBMOD   - $dff062
BLTCMOD   - $dff060
BLTDMOD   - $dff066
```

Każdy z portów ABC ma alternatywny port abc, w którym dane zawarte w ABC są negowane..

Daną operację przeprowadzaną na Blitterze zapisuje się równaniem i za pomocą niego wylicza się, które bity należy zapalić w komórce odpowiadającej za operację jaką wykona Blitter. Komórką tą jest BLTCON0 i kolejno bity odpowiadają za:

BLTCON0 - $dff040 (słowo)

bity	funkcja	opis
15	ASH3	bity ASH odpowiadają za przesunięcie
14	ASH2	porty źródłowego A o
13	ASH1	odpowiednią ilość bitów (od 0
12	ASH0	do 15)
11	USEA	czy jest używany port A
10	USEB	czy jest używany port B
9	USEC	czy jest używany port C
8	USED	czy jest używany port D
7-0	LF7-0	operacja logiczna

Tablica 1.

LF7 = ABC	LF6 = ABc	LF5 = AbC	LF4 = Abc
LF3 = aBC	LF2 = aBc	LF1 = abC	LF0 = abc

Po ustawieniu wszystkich komórek należy uaktywnić Blitter wpisując długość obszaru na jakim chcemy wykonać operację logiczną do komórki:

```
BLTSIZE   - $dff058 (słowo)
```

Bity 15-6 odpowiadają za wysokość obszaru poddawanego operacji i bity 5-0 za jego szerokość w słowach.

Jeżeli na przykład chcemy przeprowadzić operację na obszarze pamięci o wysokości 32 linii i szerokości 64 pixeli (4 słowa) to do BLTSIZE wpisujemy $0084 (32 mnożymy przez 64 i 64 dzielimy przez 16, następnie otrzymane wartości dodajemy. Wynika z tego, że najmniejszym obszarem roboczym dla Blittera jest słowo.

Aby wykonać jakąkolwiek operację musimy sobie ułożyć równanie względem którego będzie postępować nasz blitter.

Musimy wiedzieć, że A+a daje obszar samych jedynek (tak jak w prawdopodobieństwie: zbiór A i przeciwny do niego A' dają zbiór wszystkich zdarzeń elementarnych) oraz to, że możemy wykonywać tylko operacje dodawania i mnożenia (+,*).

Przykład 1.

Jeżeli chcemy przenieść obszar pamięci to budujemy następujące równanie:

D = A * (B+b) * (C+c)

Czyli przeznaczenie wynosi: wartości pobierane z portu A mnożone razy 1 (B+b = C+c = 1) czyli D=A. Następnie wyliczamy wzór i otrzymujemy:

D = ABC + ABc + AbC + Abc

Patrzymy teraz, które bity należy ustawić (Tablica 1) i otrzymujemy, że bity LF7, LF6, LF4, LF3 oraz musimy ustawić bity USEA i USED (gdy chcemy używać tylko portu A i D). Bity ASH3-0 ustawiamy w zależności od tego czy chcemy aby obszar kopiowany był przesunięty o odpowiednią (0-15) ilość bitów, jeżeli chcemy go tylko skopiować to ustawiamy je na 0. Czyli do BLTCON0 wpiszemy wartość $09f0-%0000100111110000.

Blitter posiada również drugą komórkę kontroli:

BLTCON1 - $dff042 (słowo)

bity	funkcja	opis
15-12	BSH3-0	przesunięcie portu źródłowego B
11-5	nieużywane	
4	EFE	bit funkcji Exclusive Fill
3	IFE	bit funkcji Inclusive Fill
2	FCI	bit Fill Carry In
1	DESC	tryb Descending
0	LINE	kreślenie linii

Bity BSH3-0 odpowiadają za przesunięcie danych pobieranych z portu B podobnie jak z A. Tylko port C nie ma przesunięcia co oznacza, że może być używany tylko do danych nie potrzebujących przesuwania.

Bity EFE, IFE, FCI są używane wtedy gdy wypełniamy obszary pamięci za pomocą Blittera natomiast Blitter LINE - gdy kreślimy linie (zostaną one bliżej omówione w artykułach o tworzeniu programów demonstracyjnych - grafice wektorowej). Bit DESC jeżeli jest wyłączony to Blitter pracuje normalnie,

a jeżeli jest ustawiony to Blitter wykonuje swoją pracę do tyłu, przenosi obszary pamięci od tyłu, wypełnia, itp. co jest przydatne jeżeli obszar przenoszony pokrywa się z obszarem do którego przenosimy dane i nie chcemy, aby coś zostało zamazane.

Blitter posiada jeszcze dwie ważne komórki:

```
BLTAFWM    - $dff044
BLTALWM  -   $dff046
```

Są to maski pierwszego i ostatniego słowa z portu A (tj. wartość z którą nastąpi logiczny AND początku i końca każdej linii pobieranej z portu A).

Ważną rzeczą - o której nie należy zapominać - jest sprawdzenie przed wywołaniem Blittera czy nie wykonuje on jeszcze poprzedniej pracy (jest on koprocesorem i pracuje niezależnie od procesora). Sprawdzenie polega na teście 6 bitu komórki DMACONR:

```
WaitBlitterFinnish: btst    #6,$dff002
                    bne.s   WaitBlitterFinnish
                    ...
```

A teraz już przejdźmy do konkretnych przykładów:

Przykład 2.

Przeniesienie obrazu 320*256 pixeli spod adresu $40000 pod adres $60000:

```
move.l #$40000,BLTAPT       ; źródło
move.l #$60000,BLTDPT       ; przeznaczenie
move.w  #$ffff,BLTAFWM
move.w #$ffff,BLTALWM       ; maski aby nie obciąć
                            ; pierwszych słów
move.w #$09f0,BLTCON0       ; operacja D=A
move.w #$0000,BLTCON1       ; żadnej operacji
move.w #$0000,BLTAMOD       ; modulo portu A
move.w #$0000,BLTDMOD       ; modulo portu D
move.w #$4014,BLTSIZE       ; 256*64+(320/16)
```

W tym momencie Blitter rozpocznie pracę przenoszenie danych.

Przykad 3.

Stawianie Bobsów. Aby uzyskać efekt wielu obiektów na ekranie latających po tle należy je na to tło „narzucać", czyli najpierw usunąć pewien obszar tego tła (odpowiadający kształtem obiektowi), a następnie narzucić obiekt w to miejsce. Do tego celu musimy wykorzystać trzy porty Blittera: w porcie A będziemy mieli dane dla obiektu (obiekt o wymiarach np. 16*10) ułożone w następujący sposób: 1 linia danych obiektu, słowo o wartości 0, 2 linia danych, słowo 0, itd. W porcie B umieszczamy dane maski (taki sam sposób ułożenia danych jak obiekt), w porcie C umieszczamy adres pamięci, w który narzucamy obiekt (logiczne AND z mask i logiczne OR z obiektem) a w porcie D umieszczamy obszar do którego odsyłamy wynik (w naszym przypadku będzie to ta sama wartość jak w porcie C, gdyż to właśnie tam obiekt ma zostać narzucony). Słowo o wartości 0 po każdej linii danych dla obiektu i maski umieszczone jest dlatego, że obiekty będziemy przesuwać o jeden pixel (używane bity ASH3-0 i BSH3-0) i powodowało by to znikanie grafiki z prawej strony obiektu. Zakładamy, że nasz obiekt (wymiary 16*10) mamy umieszczony pod adresem $30000, jego maskę (takie same wymiary) pod adresem $31000, ekran (320*256), na który narzucamy obiekty pod adresem $40000, a obiekt chcemy narzucić w obszar o współrzędnych 98,13.

Na początku musimy obliczyć pod jakim adresem znajdzie się obiekt: ma on być w linii 13 (długość linii 40 bajtów) czyli do $40000 dodajemy 13*40=$208 otrzymujemy adres $40208. Teraz musimy obliczyć jego współrzędną poziomą (pozycję dzielimy przez 16 i następnie mnożymy przez dwa (pamiętajmy, że Blitter pracuje na słowach!) reszta jest przesunięciem ASH i BSH). W naszym przypadku otrzymamy wartości 12 i resztę 2. Wynik dzielenia dodajemy do adresu i otrzymujemy $40214.

Następnie musimy wykonać operację na Blitterze:

D = ((A+a) * b * C) + (A * (B+b) * (C+c))

Pierwsza część to AND tła (C) z inwersją maski (b), po której pozostanie tło wokół obiektu, a druga część to dodanie (OR) obiektu. Po wyliczeniu otrzymujemy:

```
D = AbC + abC + ABC + AbC + ABc + Abc
     LF5   LF1   LF7   LF5   LF6   LF4
```

W wyniku otrzymujemy dla BLTCON0 $0FF2 - używamy portów A B C i D a następnie ustawiamy bity ASH na 2 - wartość przesunięcia obiektu, otrzymujemy $2ff2 a komórka BLTCON1 będzie tylko zawierała przesunięcie dla maski czyli $2000.

```
move.l #$30000,BLTAPT       ; adres obiektu
move.l #$31000,BLTBPT       ; adres maski
move.l #$40214,BLTCPT       ; adres obszaru gdzie
                            ;  umieszczamy obiekt
move.l   #$40214,BLTDPT
move.w    #$ffff,BLTAFWM
move.w #$ffff,BLTALWM       ; maski aby nie obciąć
                            ;  pierwszych słów
move.w #$2ff2,BLTCON0       ; kod operacji
move.w    #$2000,BLTCON1
move.w #$0000,BLTAMOD       ; modulo dla portu A
                            ;  (obiekt)
move.w #$0000,BLTBMOD       ; modulo dla portu B
                            ;  (maska)
move.w #$0024,BLTCMOD       ; modulo dla portu C
                            ; i D wynika z tego, że
move.w #$0024,BLTCMOD       ; obiekt ma dugość 1
                            ; słowa i 1 słowa
                            ;  odstępu
move.w #$0282,BLTSIZE       ; szerokość obiektu
                            ; wynosi 1 sowo, ale
                            ;  mamy
                            ; jeszcze jedno słowo
                            ;  rezerwy (dlatego 2)
```

Przykład 4.

Scroll. Przesuwający się obszar pamięci o dowolną ilość bitów to nic innego jak kopiowanie pamięci z pewnego miejsca pamięci w to samo z przesunięciem portu A. Należy jednak zauważyć, że Blitter w „normalnym" trybie pracy przesuwa w prawo a na ogół scroll'e są robione w lewo. Aby uzyskać efekt scroll'a w lewo musimy wykorzystać do tego celu bit DESC z komórki BLTCON1.

Wykonamy scroll pamięci pod adresem $40000 o wymiarach 320*100 o 3 bity w lewo.

W tym celu musimy obliczy adres końca scrollowanego obszaru (40 bajtów szerokości * 100 linii wysokości daje nam $fa0 czyli ostatnie słowo tego obszaru jest pod adresem $40f9e).

```
move.l #$40f9e,BLTAPT        ; adres końca obszaru
                             ;   scrollowanego
move.l #$40f9e,BLTDPT        ; adres końca obszaru
                             ;   scrollowanego
move.w #$39f0,BLTCON0        ; operacja. kopiowanie
                             ; i przesunięcie o 3
move.w #$0002,BLTCON1        ; tryb Descending
                             ; (od tyu)
move.w #$ffff,BLTAFWM
move.w #$ffff,BLTALWM        ; maski aby nie obciąć
                             ;   pierwszych słów
move.w #$0000,BLTAMOD        ; modulo zerowe
move.w #$0000,BLTAMOD        ; modulo zerowe
move.w #$1914,BLTSIZE        ; obszar o wymiarach
                             ; 320 ($14 sw) na 100
                             ; linii ($64)
```

Przykad 5.

Wypełnianie. Blitter może wypełniać ograniczone przestrzenie. Wypełnia on linie od pierwszego zapalonego punktu do następnego, tj. rozpoczyna wypełniać gdy napotyka punkt nieparzysty, a kończy gdy napotka parzysty lub koniec linii.

Wypełnianie to nic innego, jak kopiowanie obszaru z jednego miejsca w drugie z włączonym bitem IFE (Inclusive Fill Enable). Jeżeli chcemy np. wypełnić pewne obszary zawarte na przestrzeni takiej, jak w przykładzie 2 (umieszczonej pod adresem $40000 to musimy ją przenieść spod tego adresu pod ten sam z ustawionym bitem IFE w BLTCON1).

```
move.l #$40000,BLTAPT        ; źródło
move.l #$40000,BLTDPT        ; przeznaczenie
move.w #$ffff,BLTAFWM
move.w #$ffff,BLTALWM        ; maski aby nie obciąć
                             ;   pierwszych słów
move.w #$09f0,BLTCON0        ; operacja D=A
move.w #$0008,BLTCON1        ;   operacja wypeniania
                             ;   (IFE=1)
move.w #$0000,BLTAMOD        ; modulo portu A
move.w #$0000,BLTDMOD        ; modulo portu D
move.w #$4014,BLTSIZE        ; 256*64+(320/16)
```

Myślę, że to powinno wystarczyć czytelnikom do rozpoczęcia pracy z Blitterem i zmuszania go do tego co my chcemy aby zrobił - bo to w końcu my mamy mózgi...

Marcin „Duddie" Dudar

JAK ZROBIĆ DEMO (5)

W poprzednim odcinku (patrz str. 8) zajmowaliśmy się bliżej procedurami do tworzenia płynących napisów, czyli tzw. scroll'ami. Teraz przedstawię dalszy ciąg tego zagadnienia, opisując scroll o podwójnej wielkości znaków. Łatwo taką procedurę rozpoznać w programie demonstracyjnym; charakterystyczna jest bowiem wspomniana powyżej wielkość liter - każda składa się z czterech, uprzednio odpowiednio przedefiniowanych, „zwykłych" znaków.

Aby lepiej zrozumieć zasadę działania całej procedury, dobrze by było przedstawić zasadę przedefiniowania oryginalnego zestawu znaków na takie, które potem mogły by zostać użyte przez naszego scroll'a. Jak zapewne Wam wiadomo, generator znaków (czyli zbiór danych opisujących kształt liter na potrzeby komputera) zajmuje w pamięci dokładnie dwa kilobajty, czyli $0800 (w zapisie heksadecymalnym).

Dla przypomnienia zobaczmy, jak ta pamięć została wykorzystana w przypadku standardowego zestawu znaków (tego zapisanego w pamięci ROM). Całość daje się łatwo podzielić na 4 bloki po 64 znaki (jak pamiętamy, generator jednorazowo opisuje 256 znaków):

1. $0000-$01FF 64 znaki podstawowe
2. $0200-$03FF 64 znaki, tzw. „z Shiftem"
3. $0400-$05FF 64 znaki podstawowe w inverse-video (negatywy)
4. $0600-$07FF 64 znaki „Shiftem" w inverse-video

Nie trudno zauważyć, że na nasze potrzeby całkowicie wystarczą pierwsze 64 znaki podstawowe (tzn. 64 pierwsze kody edkranowe, od $00 do $3F). Zawierają się w nich wszystkie litery i cyfry oraz kilka znaków specialnych, jak znaki interpunkcyjne, nawiasy itp. Dla dociekliwych krótki program w BASICu, który to wyjaśni:

```
10 ad=1024:for t=0 to 63:poke ad+t,t:next t
```

Przy pomocy tej jednej lini BASICa możemy wydrukować w górnych liniach ekranu pełny zestaw „znaków podstawowych" posługując się ich kodami ekranowymi. W naszym 2x2 scroll'u niewykorzystane ćwiartki generatora znaków użyjemy do zdefiniowania pozostałych części każdego z dużych znaków. Definiując je przyjmijmy zasadę takiego rozłożenia znaków:

```
AB
CD
```

Jak widać, każdy z dużych znaków składać się będzie z czterech odpowiednio zdefiniowanych znaków - po jednym z każdej z wspomnianych powyżej 4 grup 64 znakowych. W praktyce będzie to wyglądać tak, że element A będzie znakiem z pierwszej grupy, B - z drugiej a C i D z trzeciej i czwartej. I tak, np. litera „a", która posiada kod ekranowy równy 1, w swojej „dużej" wersji składać się będzie z czterech znaków o następujących kodach: A - 1; B - 1+64 ($41 hex); C - 1+128 ($81 hex) i D - 1+172 ($C1 hex). Nietrudno zauważyć, że każdy następny kod znaku jest po prostu efektem dodania $40 do kodu znaku poprzedniego, co będzie pewnym ułatwieniem przy pisaniu procedury. Gotowe, w ten sposób zdefiniowane znaki najwygodniej jest po prostu „wyciąć", z któregoś z dostępnych programów demonstracyjnych. W dobrym tonie jest potem poinformowanie w tekście, kto jest prawdziwym autorem użytych przez nas znaków.

Co zrobić, gdy nie mamy pod ręką żadnego dema z odpowiednio zdefiniowanymi znakami lub, po prostu, niepotrafimy ich z tego programu „wyciągnąć"? Pozwolę sobie zaprezentować niewielką procedurę, która odpowiednio użyta w naszym programie „powiększy" standardowy zestaw znaków do dwukrotnie większych wymiarów w sposób odpowiedni dla naszego scroll'a. Aby umożliwić łatwiejsze wykorzystanie procedury, została ona napisana pod Turbo-Assembler'em:

```
                       *= $1000
CHARBANK   =$D000      ;położenie
                       ;generatora znaków
CHRPLACE   =$2000      ;adres dla
                       ;powiększonych znaków
DATAS      =$0340      ;miejsce na dane dla
                       ;obliczeń (3 bajty)
           SEI
           LDA #$33
           STA $01     ;ustawienie konfiguracji
                       ;ROM
           LDX #$00
LOOP1      LDA #$00
           STA $FC
           TXA
           ASL A
           ROL $FC
           ASL A
           ROL $FC
           ASL A
           ROL $FC
           STA $22
           STA $FB
           LDA $FC
           STA $23
           CLC
           ADC #CHARBANK
           STA $FC
           TXA
           PHA
           LDY #$00
LOOP2      LDA $22
           STA $20
           LDA $23
           STA $21
```

```
        LDA ($FB),Y
        LDX #$03
LOOP3   ASL A
        PHP
        ROL DATA
        PLP
        ROL DATA
        DEX
        BPL LOOP3
        LDX #$03
LOOP4   ASL A
        PHP
        ROL DATA+1
        PLP
        ROL DATA+1
        DEX
        BPL LOOP4
        STY DATA+2
        TYA
        ASL
        TAY
        LDA $21
        CLC
        ADC #CHRPLACE
        STA $21
        LDA DATA
        STA ($20),Y
        INY
        STA ($20),Y
        DEY
        LDA DATA+1
        INC $21
        INC $21
        STA ($20),Y
        INY
        STA ($20),Y
        LDY DATA+2
        INY
        CPY #$04
        BNE LOOP2
LOOP5   LDA $22
        STA $20
        LDA $23
        STA $21
        LDA ($FB),Y
        LDX #$03
        ASL A
        PHP
        ROL DATA
        PLP
        ROL DATA
        DEX
        BPL LOOP5
        LDX #$03
LOOP6   ASL A
        PHP
        ROL DATA+1
        PLP
        ROL DATA+1
        DEX
        BPL LOOP6
        STY DATA+2
        TYA
        ASL
        SEC
        SBC #$04
        TAY
        LDA $21
        CLC
        ADC #$04
        ADC #CHRPLACE
        STA $21
        LDA DATA
        STA ($20),Y
        INY
        STA ($20),Y
        DEY
        INC $21
        INC $21
        LDA DATA+1
        STA ($20),Y
        INY
        STA ($20),Y
        LDY DATA+2
        INY
        CPY #$08
        BEQ LOOP7
        JMP LOOP2
LOOP7   PLA
        TAX
        INX
        CPX #$40
        BCS LOOP8
        JMP LOOP1
LOOP8   LDA #$37
        STA $01       ;standardowa
                      ;konfiguracja ROM
        CLI
        RTS           ;powrót z procedury
```

Po wywołaniu tej procedury uzyskamy odpowiednio powiększone znaki, które zostaną umieszczone w pamięci od adresu $2000 do $27FF, tak jak zostało to zdefiniowane przy etykiecie CHRPLACE. Łatwo zauważyć, że procedura ta może również z powodzeniem powiększać także inne znaki (tzn. oprócz tych zdefiniowanych standardowo). Aby tą operację przeprowadzić, należy zmodyfikować adres przypisany etykiecie CHARBANK tak, aby wskazywał na inne, uprzednio przez nas umieszczone dane.

Posiadamy już prawidłowo zdefiniowany zestaw znaków i możemy przystąpić do napisania procedury samego scroll'a, który będzie przewijał tekst w dwóch górnych liniach ekranu.

Ale o tym w następnym numerze...

Paweł Sołtysiński (Polonus/Padua)

ASSEMBLER 6510 - lekcja 4

Po raz drugi w tym numerze CZEŚĆ! Nareszcie dostałem kilka listów, które upewniły mnie, że prowadzenie tego kącika jest celowe. Jedni chwalą, drudzy ganią, ale ogólnie pomysł się Wam podoba.

Nie wdając się w dłuższe komentarze zaczynamy dalej „gryźć maszynówkę".

JMP (Jump) - skok

Wszystkie rozkazy skoków nie zmieniają ustawienia bitów w rejestrze znaczników. Aby oszczędzić trochę miejsca będę pomijał więc „rozpiskę" flagów.

ROZKAZ	KOD	CYKLE
JMP $nnnn	4C	3
JMP ($nnnn)	6C	5

Użycie rozkazu JMP powoduje skok do adresu podanego w argumencie. Jeśli np. wykonamy rozkaz:

```
JMP $C000
```

to mikroprocesor zacznie wykonywać program umieszczony w pamięci od adresu C000. Rozkaz JMP ($nnnn) działa tak samo jak poprzedni tyle, że w inny sposób oblicza adres - pobiera go z komórek o adresie $nnnn, np.

```
JMP ($C000)
```

pobierze adres znajdujący się w komórkach C000 oraz C001 i dopiero potem w niego skoczy.

JSR (jump to subroutrine) - skok do podprogramu

ROZKAZ	KOD	CYKLE
JSR $nnnn	20	6

Rozkaz JSR działa podobnie jak JMP tyle, że daje większe możliwości. Przed wykonaniem skoku na tzw. stos, odkładany jest adres następnej instrukcji po JSR. Daje to nam możliwość „bezbolesnego" powrotu (nie trzeba pamiętać adresu - robi to za nas komputer) do wcześniej wykonywanych czynności.

Zobaczycie w przyszłości jak przydatny jest ten rozkaz.

RTS (return from subroutine) - powrót z podprogramu

ROZKAZ	KOD	CYKLE
RTS	60	6

Uzupełnieniem rozkazu JSR jest właśnie RTS. Powoduje przekazanie sterowania w adres zdjęty ze stosu (zapisany wcześniej np. przez instrukcję JSR). Jest to więc również rozkaz skoku. Np:

```
1000 JSR $1300     ; skok do podprogramu pod $1300
1003 JSR $1500     ; skok do podprogramu pod $1500
1006 JSR $1300     ; ponowny skok pod $1300
1006 BRK
1300 —             ; podprogram
     ---
     ---
     RTS           ; powrót
1500 —             ; podprogram
     ---
     ---
     RTS           ; powrót
```

Najpierw uruchomi się podprogram leżący pod adresem $1300, a potem ten pod $1500. Po ponownym wykonaniu skoku pod $1300 komputer powróci do adresu $1006 (kolejny rozkaz po JSR). Kolejną grupą rozkazów są tzw. skoki warunkowe. Ich wykonanie jest uzależnione od tego czy dany warunek jest czy nie jest spełniony.

BEQ (branch if equal to zero) - skocz gdy zero

ROZKAZ	KOD	CYKLE
BEQ	F0	3+1

Gdy bit Z będzie ustawiony (Z=1) nastąpi skok. Np.

```
1000 LDX #$00      ; ustawienie rejestru pomocniczego
1002 LDA #$00      ; ustawienie rejestru testowanego
1004 BEQ $1008     ; sprawdzenie warunku i ewentualny
                   ; skok
1006 LDX #$01      ; zmiana rejestru pomocniczego
1008 BRK           ; koniec
```

W naszym przykładzie rejestr X pełni rolę testującą działanie rozkazu BEQ. Jeśli Z=1 (A=0) nastąpi skok pod adres $1008 czyli rejestr X będzie zawierał wartość 0. Jeśli zmienimy wartość ładowaną do akumulatora na jakąkolwiek inną (różną od zera) to Z=0 i skok nie nastąpi. Komputer wykona instrukcję leżącą zaraz po BEQ. W naszym przykładzie jest to ładowanie do X wartości 1. Kontrolując więc zawartość rejestru X możecie sprawdzić działanie rozkazu BEQ.

Jeśli uważnie przyglądaliście się naszemu przykładowi to na pewno zainteresowała was postać zapisu argumentu (adresu) dla rozkazu BEQ. Monitor wydrukował na ekranie adres szesnastobitowy (cztery cyfry), a przecież rozkaz zajmuje tylko dwa bajty. Jak to się dzieje? Otóż wszystkie rozkazy skoków warunkowych NIE ZAPISUJĄ adresu w postaci gotowej lecz OBLICZAJĄ go w zależności od swojego położenia w pamięci.

Jak pamiętacie komputer cały czas w specjalnym rejestrze PC (licznik programu) pamięta adres wykonywanej instrukcji. Po wykonaniu każdego rozkazu licznik ten zwiększa się o ilość bajtów zajmowanych przez daną instrukcję.

Rozkazy skoków warunkowych działają właśnie na tym liczniku. Zobaczcie, jak wygląda zapis rozkazu BEQ w kodzie szesnastkowym. W naszym przykładzie jest to: F0 i 02. Wartość $F0 odpowiada kodowi rozkazu, natomiast 02 to nasz adres. Komputer zadziała w

następujący sposób: po wykonaniu rozkazu BEQ zwiększył PC na $1006 (następny rozkaz), teraz dodał do tej wartości argument ($02) i wynik wpisał z powrotem do PC ($1008) - nastąpił skok. Jeśli więc warunek jest spełniony do PC zostaje dodany argument, jeśli nie to argument jest pomijany. Takie zachowywanie się rozkazów skoków jest bardzo pomocne przy pisaniu procedur relokowalnych czyli takich, które mogą leżeć w dowolnym miejscu pamięci. Wykonajcie teraz operację na monitorze:

```
T 1000 1010 3F10
D 3F10
```

Mimo, że program znalazł się w zupełnie innym obszarze pamięci dalej jest poprawny: skok BEQ sam się „przeliczy". Jedyną wadą jest zapis argumentu na jednym bajcie. Przez to skoki można wykonywać jedynie w zakresie -128 do +127 bajtów od położenia rozkazu Branch.

BNE (branch if not equal to zero) - skok gdy rezultat niezerowy

ROZKAZ	Kod	Cykle
BNE	D0	3+1

Gdy zawartość znacznika Z=0 następuje skok.

BCC (branch if carry clear) - skok gdy przeniesienie skasowane

ROZKAZ	Kod	Cykle
BCC	90	3+1

Skok następuje wtedy gdy C=0

BCS (branch if carry set) - skok gdy przeniesienie ustawione

ROZKAZ	Kod	Cykle
BCS	B0	3+1

Jeśli C=1 to nastpuje skok.

BPL (branch if plus) - skok gdy rezultat nieujemny (N=0)

ROZKAZ	Kod	Cykle
BPL	10	3+1

BMI (branch if minus) - skok gdy rezultat ujemny (N=1)

ROZKAZ	Kod	Cykle
BMI	30	3+1

Aby zrozumieć działanie rozkazów BPL i BMI w podanym wcześniej przykładzie zmieniajcie rozkazy skoków i obserwujcie zachowanie się rejestru X.

BVC (branch if overflow clear) - skok gdy nadmiar skasowany (V=0)

ROZKAZ	Kod	Cykle
BVC	50	3+1

BVS (branch if overflow set) - skok gdy nadmiar ustawiony (V=1)

ROZKAZ	Kod	Cykle
BVS	70	3+1

6510 ma kilka rozkazów dzięki którym programista może bezpośrednio ingerować w zawartość rejestru znaczników. Oto kilka z nich:

CLV (clear overflow) - skasuj nadmiar

N	V	D	I	Z	C
-	0	-	-	-	-

ROZKAZ	Kod	Cykle
CLV	B8	2

SEC (set carry) - ustaw przeniesienie

N	V	D	I	Z	C
-	-	-	-	-	1

ROZKAZ	Kod	Cykle
SEC	38	2

CLC (clear carry) - skasuj przeniesienie

N	V	D	I	Z	C
-	-	-	-	-	0

ROZKAZ	Kod	Cykle
CLC	18	2

Na koniec jak zwykle małe zadanko dla pracowitych (tym razem od razu z rozwiązaniem dla leniuszków). Ostatnio dodawaliśmy do siebie dwie liczby. Obie z nich były jednak tylko ośmiobitowe. Jak dodać dwie liczby szesnastobitowe?

```
1000    LDX  #$00
        LDA$1100
        CLC
        ADC $1102
        STA $1104
        LDA $1101
        ADC $1103
        STA $1105
        BCC $1018
        INX
1018    BRK
```

Programik dodaje dwie liczby znajdujące się w adresach $1100 i $1102, a wynik odkłada pod $1004. Jeżeli wynik będzie zbyt duży (liczba nie mieszcząca się w szesnastu bitach) program zareaguje na to sygnalizując błąd przez wpisanie do rejestru X wartości 1.

Sambor Kufma

** MAGIĄ I MIECZEM ** czyli RATUUUNKUUUUU!

Gry przygodowe stanowią dość pokaźny procent wszystkich gier dostępnych na nasze maszyny. Ostatnimi czasy odnotowujemy znaczny wzrost zainteresowania tym gatunkiem. Być może ludziom znudziły się już bezsensowne strzelaniny, które polegają jedynie na szybkim naciskaniu fire czy gry sportowe, w których na ogół chodzi o połamanie joystick'a.

Gry przygodowe (tzw. Adventures) przenoszą nas w najróżniejsze bajeczne światy, których rzeczywistość jest ograniczona jedynie wyobraźnią autora. Przed graczem stawiane są różne problemy, zagadki, które korzystając z własnej wyobraźni musi pokonać, rozwiązać po to aby osiągnąć upragniony cel. Takie gry potrafią przyciągnąć do komputera na całe tygodnie. Człowiek potrafi zapomnieć o tej prawdziwej rzeczywistości delektując się światem stworzonym przez programistów. Często jednak gra taka potrafi przyprawić o atak szału, który może zakończyć się nawet rzuceniem komputera o ścianę (sam byłem kiedyś bliski tego). Zapobieganiu tego rodzaju zachowaniom ma służyć ta właśnie rubryka. Powstała ona na wyraźne życzenie czytelników.

Przyszło do nas kilka listów z pytaniami dotyczącymi gier przygodowych i z prośbą o stworzenie kącika pomocy zagubionym. Często bowiem zdarza się, że w pewnym momencie utknęliśmy gdzieś i nie wiemy co i jak dalej zrobić aby rozwiązać kolejną zagadkę. Bezowocne siedzienie przy klawiaturze bardzo męczy i irytuje. Od tej pory więc jeśli gdzieś stanęliście piszcie do nas!

Aby otrzymać pomoc wystarczy przysłać list z podanym tytułem gry i opisaną sytuacją, w której utknęliście. Jeśli redakcja będzie znała odpowiedź to postaramy się ją zamieścić. Jeśli jednak będziemy wiedzieli tyle samo co pytający to... wystosujemy prośbę do czytelników, aby jeśli znają rozwiązanie danego problemu napisali o tym do redakcji. W ten sposób mamy nadzieję szybko pomagać naszym zagubionym czytelnikom.

RATUUNKUUU!

1. Jak znaleźć Sword Master'a w grze The Secret Of Monkey Island. Jestem już wyćwiczonym fechmistrzem. Co dalej? Maciek Morawski z Katowic.

Aby znaleźć Sword Master'a musisz pójśćdo lasu (Fork) i znaleźć znak „Uwaga rozpadlina!" Teraz wystarczy uderzyć w znak łopatą i... do następnej zagadki!

2. Jak robić czary w grze Elvira? Znalazłem książkę czarów lecz dalej nie potrafię nic zrobić. Bartosz ...? z Opola (strasznie niewyraźnie piszesz Bartek).

Niestety nie mogę Ci w tym miejscu odpowiedzieć na to pytanie. Przepisy na czary są oryginalnie zamieszczone w instrukcji do gry. Jest ich bardzo dużo i nie zmieściłyby się tutaj. Na jednym z kolejnych PDP zamieścimy gotowy zbiór tekstowy ze spisem wszystkich potrzebnych składników. Wystarczy zamówić nasze PD Pack'i.

3. Jak przejść granice w grze Indiana Jones III? Boksowanie się ze strażnikami jest nie do przejścia! Jacek Królak z Warszawy.

Widzę Jacek, że nie czytałeś poprzednich numerów „64 + 4"! W marcowym wydaniu są zamieszczone podpowiedzi do tej gry. Postaraj się zdobyć ten numer. Abyś jednak nie musiał zbyt długo czekać podpowiem Ci, że przechytrzyłeś sam siebie dając Niemcom fałszywy dziennik ojca! Jedyne co pozostało Ci do zrobienia to wczytać poprzednią sytuację (mam nadzieję, że regularnie wszystko zapisywałeś) i daćSS-manowi PRAWDZIWY Grail Diary.

Do zobaczenia za miesiąc.

Sambor Kuźma



PACKERY, CRUNCHERY, ARCHIEVERY
...czyli o upychaniu małe co nieco

Aby zaoszczędzić cenne miejsce na nośnikach magnetycznych ludzie wymyślili programy do kompresji danych.

Pierwszym typem takich programów są packery i crunchery. Różnią się one od siebie tylko efektywnością kompresji danych, a służą do zmniejszania długości plików. Jako metody kompresji najczęściej używa się ideę wyszukiwania podobnych ciągów bajtów (cruncher sekwencyjny), podobnych bitów w kolejno następujących po sobie bajtach (cruncher bitowy), a także pakowania następujących po sobie takich samych bajtów (cruncher znacznikowy).

Na Amidze możemy odnaleźć bardzo wiele programów kompresujących dane. Można je podzielić na dwie grupy: crunchery do danych lub programów lokowanych w pamięci pod adresami absolutnymi (programy nierelokowalne) oraz crunchery do programów relokowalnych. Do najlepszych cruncher'w możemy zaliczyć:

Nazwa Crunchera	Autor
Black&Decker Cruncher	
ByteKiller v2.0	
DragPack v1.0b	Hawk
DoubleAction	
DefJam Packer	Antiaction
MasterCruncher v3.0	M.Cremer & R.Frahm
MegaCruncher	
Power Packer v3.0a	Nico „Lone Wolf" Francois
StoneCracker	
SuperCruncher v2.7	Fast Team
Supplex Cruncher	
SyncroPack	Syncro
Time Cruncher	
Titanics v1.1	Triad
Turbo Imploder v3.0	P.Struijk & A.Brouwer

Najlepszymi programami do kompresji bloków relokowalnych są: Power Packer i Turbo Imploder. Zdecydowanie biją na głowę inne programy pod względem efektywności, a przede wszystkim pod względem czasu kompresji. Program Power Packer ma bardzo wiele dodatkowych opcji, na przykład: kompresja programów nakładkowanych (np. Sculpt 4D lub DeLuxe Paint III), kompresja danych (teksty, grafika), kompresja bardzo wielu programów bez konieczności ciągłego wybierania odpowiednich tytułów (np. kompresja całego dysku z instrukcjami do programów). Do tego programu istnieje wiele programów narzędziowych:

PPMore - do odczytywania skompresowanych tekstów
PPShow - do oglądania skompresowanych obrazków
PPAnim - do odtwarzania skompresowanych animacji

Wiele nowych programów ma wbudowane procedury do dekompresji danych skompresowanych Power Packerem np. MasterSeka może wczytywać skompresowane teksty źródłowe, ProTracker może wczytywać skompresowane moduły muzyczne i wiele innych programów. Power Packer stał się standardem.

A teraz trochę charakterystyki porównawczej.

Program kompresowany: D-Mon Professional v2

	długość	czas kompresji
Nieskompresowany	33088	
PowerPacker v3.0a	20560	13 sekund
Turbo Imploder v3.0	20024	19 sekund
DragPack v1.0b	22456	7 min. 53 sek.
Master Cruncher v3.0	20516	11 min. 5 sek.
Titanics Cruncher v1.1	22844	4 min. 7 sek.

Jak widać różnica w czasie kompresji jest znaczna przy niewielkiej różnicy długości. Program Titanics Cruncher, który może nie odniósł wielkich sukcesów jeżeli chodzi o długość programu po kompresji, ale jest za to niezastąpionym narzędziem jeżeli chodzi o programy o długości powyżej 250KB gdy nie wczytuje ich w całości do pamięci, ale dekompresuje je w czasie odczytywania z dysku. Dzięki temu możemy zaoszczędzić wiele miejsca na dysku kompresując programy, których wcześniej nie mogliśmy skompresować gdyż posiadaliśmy zbyt mało pamięci na kompresję lub dekompresję.

Program Titanics Cruncher zostanie umieszczony na jednym z dysków Public Domain. Niestety nie możemy zamieścić pozostałych kompresorów gdyż są one programami komercjalnymi.

Kolejną grupą kompresorów są **kompresory dyskowe.** Służą one do składowania części lub całego dysku w formie zbioru. Jest to użyteczne gdy na przykład chcemy przesłać komuś przez modem kilka dysków (skraca czas transmisji). Programy te wczytują do pamięci odpowiednią ilość sektorów, a następnie je kompresują a wynik jest zapisywany w postaci zbioru na dysku. A teraz mała charakterystyka porównawcza kompresorów dyskowych:

Dysk kompresowany: D-Mon Professional v2.0

Kompresor	Autor	Długość zbioru	Czas kompresji
DIMP v1.0	A.J.Brouwer	101986	21 min. 20 sek.
ZAP v1.4	Gremlin/Mayhem	120772	7 min. 11 sek.
WARP v1.	Nieznany	128818	7 min. 7 sek.
ZOOM v4.1	Olaf Barthel	102592	1 min. 32 sek.

Jak widać największy sukces jeżeli chodzi o długość zbioru odniósł DIMP. Niewiele gorszym od niego okazał się ZOOM, ale zrobił to w bardzo krótkim czasie. Program ten znajdziecie razem z Titanics Cruncer'em na kolejnym dysku Public Domain.

Następna grupa to **programy do archiwizacji**. Służą one do zapisania wielu plików jako jednego skompresowanego. W razie potrzeby możemy z tego pliku „wyciągnąć" aktualnie potrzebny nam zbiór, możemy go usunąć lub dopisać do tego pliku kilka innych zbiorów.

Na Amigę istnieje wiele programów wykonujących to zadanie jednak do testu wybrałem trzy najlepsze, gdyż tylko one w pełni odpowiadają użytkownikowi takiego komputera jak Amiga.

Program	Autor
ZOO v2.0	J. Brian Waters
PKAmigaZip v1.0	Dennis Hoffman
LhArcAmiga v0.99a	Stefan Broberg

Programy PKAZip i LhArcA są programami pracującymi na własnym zadaniu i wykorzystującymi multitasking natomiast programu ZOO używamy klepiąc dane z klawiatury w oknie CLI jako parametry dla tego programu lub używamy go spod DiskMastera, który to ma zaimplementowaną obsługę programu ZOO. Do testu tych programów użyłem dysku z programem D-Mon Professional v2.0, a test opiera się na kompresji wszystkich zbiorów znajdujących się na tym dysku. W tabelce znajdują się wyniki kompresji poszczególnych zbiorów i zysk procentowy.

Najlepszy wynik osiągnął program LhArcA pomimo tego, że nie jest to jeszcze wersja w pełni gotowa tego programu. Jest on dodatkowo kompatybilny z IBM'owską wersją programu LhArc co oznacza, że możemy sobie przenieść zbiór zarchiwizowany na IBM'ie (np. za pomocą Dos-2-Dos) i rozpakować go na Amidze. Program LhArcA znajdziecie także na dysku PD.

Do testów wykorzystano komputer Amiga 500 z pamięcią 1MB.

Marcin „Duddie" Dudar

	Nieskomp.	ZOO		LhArcA		PKAZip	
.info	50	6%	47	20%	40	12%	44
D-Fast	1920	30%	1348	40%	1166	38%	1196
D-Fast.Doc	819	36%	525	46%	444	38%	509
D-Fast.Doc.info	4851	75%	1234	85%	750	85%	740
D-Fast.info	6258	64%	2280	72%	1795	71%	1841
D-Mon	33088	26%	24326	43%	19071	42%	19232
D-Mon.Doc	10756	49%	5444	57%	4682	58%	4608
D-Mon.Doc.info	4851	75%	1237	85%	752	85%	742
D-Mon.info	15018	73%	4066	79%	3228	79%	3236
Disk.info	5242	72%	1446	84%	849	85%	815
c/EndCLI	696	14%	599	22%	548	17%	583
c/FF	3068	0%	3068	3%	3005	0%	3068
c/LoadWB	2804	32%	1910	42%	1649	41%	1678
c/PM	7624	0%	7624	2%	7515	0%	7624
libs/arp.library	17100	11%	15253	27%	12568	24%	13048
libs/diskfont.library	4964	34%	3261	46%	2694	46%	2719
libs/icon.library	5688	32%	3841	44%	3186	43%	3250
libs/info.library	16390	42%	9489	54%	7554	54%	7614
libs/req.library	14524	16%	12163	31%	10066	27%	10347
devs/system-coniguration	232	30%	163	37%	148	32%	160
l/Disk-Validator	1892	30%	1319	40%	1143	39%	1171
l/Ram-Handler	6200	40%	3705	52%	3031	51%	3166
s/startup-sequence	35%	0%	35	0%	35	0%	35
fonts/siesta/8	2904	37%	1833	44%	1634	38%	1829
fonts/siesta.font	784	90%	79	93%	55	93%	60
Ogółem	167748	37%	108465	48%	88454	43%	91839
Czas kompresji		1 min 20 s		3 min 14 s		3 min 29 s	

Przegląd gier cd.

Z-OUT

No i mamy kolejną kontynuację. Tym razem jest to druga część gry X-OUT. Typowa strzelanina, bardzo zbliżona do legendarnego R-TYPE. Tym razem gra jest bardzo starannie zaprogramowana (niesamowicie duże obiekty poruszające się w czasie jednej ramki) ze świetną grafiką i ładną oprawą muzyczną. Trzeba przyznać, że autorzy postarali się...

Gra jest trudna i bez żadnych ułatwień przeciętny gracz (również zaliczam się do tej grupy) dojdzie najdalej do trzeciego poziomu (a prawdziwa zabawa zaczyna się dopiero w następnych rundach). Zupełnie nie do przejścia jest ostatni, szósty poziom. Czasami nie wiadomo nawet w co trzeba strzelać aby pokonać przeciwnika.

Autorzy pomyśleli również o zabawie dla dwóch graczy. Do dyspozycji mamy wtedy dwa statki, które razem współdziałając mają o wiele większe szanse dojścia do końca niż pojedynczy gracz.

Legenda do gry jest niezwykle prosta. Jeden ze światów został opanowany przez siły Zła, a ty jako Dobry Obrońca musisz je pokonać. Po zakończeniu gry wszyscy ci dziękują i kolejny świat zachwycony twoją wspaniałą walką prosi cię o pomoc. Oczywiście jako Dobry Obrońca zgadzasz się na wszystko i tak zanosi się na następną kontynuację! (Y-OUT?)

Jak w każdej porządnej grze i tu są tzw. CHEAT'y dla wszystkich leniwych. Wystarczy nacisnąć klawisze J i K, aby stać się niewidzialnym dla przeciwnika. Jeśli któraś ze stref szybko nam się znudzi to przeskoczyć można za pomocą klawiszy J i 1,2,3,4,5,6 (cyfry oznaczają kolejne strefy).

Koncepcja i programowanie: Tobias Binsack
Grafika: Matthias Hauser, Thomas Klinger
Muzyka: Chris Huelsbeck i Rudolf Stember

Wydawca:	*Advantec 1990*
Ocena:	
Grafika:	*9*
Muzyka:	*8*
Animacja:	*10*
Pomysł:	*2*
Przyjemność	*7*
Ogólnie:	*7.2 DUŻY PLUS*

FUTURE SHOCK

Future Shock to kolejna gra platfomowa nie wyróżniająca się niczym spośród setek jej podobnych.

Zostałeś uwięziony w czasach prehistorycznych-planowałeś spędzić miłe wakacje. Odkryłeś, że skończyły Ci się kryształy litu, które jak powszechnie wiadomo są źródłem napędu do wszelkich maszyn czasu. W tym momencie zaczyna się twoje zadanie - musisz znaleźć odpowiednią liczbę kryształów.Oczywiście przeszkadzać Ci w tym będą różne wrogo do ciebie nastawione zwierzęta. Ponieważ lit jest dość rzadki musisz odwiedzić kilka epok czasu, aby móc w końcu powrócić do domu.

Jak widzicie scenariusz gry jest standardowy. Taka sama jest i sama gra. Gracz kieruje małym stateczkiem i przebijając się przez gąszcz przeciwników stara się znaleźć kryształy litu.

Osobiście dziwię się, że takie gry jeszcze powstają. Kto to kupuje? Kilka lat temu takie coś byłoby ciekawe, ale teraz... Jedyne co mi się podobało w grze to dość znośna grafika. Zaskoczony byłem jednak słabą muzyką. Jeroen Soede (sławny niegdyś Soedesoft) pisał bardzo ładne muzyczki na C-64. Czyżby Amiga wpływała destruktywnie na muzyków?

Programowanie:	*Maurice i Raymond Penner*
Grafika:	*Erwin Penders*
Muzyka:	*Jeroen Soede*
Ocena:	
Grafika:	*7*
Muzyka:	*5*
Animacja:	*5*
Pomysł:	*3*
Przyjemność:	*4*
Ogólnie:	*4.8 MAŁY PLUS*

Formuła #1 3D

Program	*2*
Grafika	*2*
Muzyka	*0 (właściwie należałoby dać -20)*
Animacja	*3*
Przyjemność	*0*

Nazwa gry nie jest zbyt zachęcająca, gdyż każdy (nawet skończony idiota - czytaj autor tej gry) może się zorientować, że jest to stary pomysł (wyścigi samochodowe) wykonany w starej formie (widok z kabiny kierowcy), ale na pewno nie każdy może się domyślić po tytule z jakim „dnem" się zetknie. Widziałem już gry ciekawe, idiotyczne, męczące, ale nigdy nie widziałem TAAAAKIEGO DNA, w które nie można grać!

Ta gra jest po pierwsze wykonana strasznie. Jak na rok 1991 to animacja wektorowa robiona co kilkanaście „ramek" jest trochę za wolna, a obiekty, z którymi mamy kontakt wzrokowy są tak prymitywnie zaprojektowane, że możemy się domyślić wszelkiego ich przeznaczenia za wyjątkiem tego właściwego. Muzyk, który tworzył muzykę do tej gry był chyba głuchy i ślepy - bo gdyby był tylko głuchy to mógłby zorientować się po rozłożeniu dźwięków na pięciolinii (czy w pattern'ach NoiseTrackera?), że coś z tą muzyką jest nie tak. Na szczęście grającego (a właściwie wczytującego ten „ciekawy" program) muzyka leci tylko na początku i można ją szybko przerwać. Za to nie można wyłączyć dźwięków towarzyszących jeździe samochodem (powiedzmy, że jest to jazda i jest to samochód), a są tak okrutne, że lepiej nie wspominać.

Ogólnie rzec biorąc tylko prawdziwy miłośnik programów niepotrzebnych, bzdurnych i idiotycznych zachowa sobie tę grę na pamiątkę. Można ją sobie też wczytać od czasu do czasu i stwierdzić jakim to jest się niesamowitym programistą, grafikiem i muzykiem w jednej osobie w porównaniu z autorami gry (choćby nic się nie robiło).

S.K.

Własne demo - Amiga - cd.

„Programista nigdy nie ma czasu" - oto jedno z podstawowych praw Murph'iego. Nie chodzi tu oczywiście o ten czas, który zużywają na pisanie programów, ale o czas ich wykonywania. Dlatego też wszystkimi możliwymi sposobami starają się go zaoszczędzić. Ale zacznijmy od początku...

Obraz, który możemy oglądać na naszych monitorach, jest generowany przez komputer 50 razy na sekundę. Co 1/50 sekundy zaczyna być rysowany liniami poziomymi zwanymi RASTRAMI, począwszy od lewego górnego rogu ekranu. Aktualnie wyświetlaną pozycję możemy kontrolować przy pomocy rejestrów VPOSR=$0004 oraz VHPOSR=$0006. Amiga posiada $138 rastrów. Gdy zostaną one wszystkie wyświetlone następuje przejście wiązki elektronów z prawego dolnego rogu ekranu do lewego górnego. Podczas tego przejścia zostaje wygenerowane przez komputer tzw. „przerwanie wygaszania pionowego: VBLK" podczas którego wykonywane są niezbędne operacje potrzebne do prawidłowej pracy systemu. Użytkownik może w łatwy sposób wtargnąć w ten mechanizm, przerwać go zlecając komputerowi dodatkowe zadania. Po wykonaniu procedur przerwania obraz wyświetlany jest od nowa. Czas potrzebny na pełne wyświetlenie ekranu nazywa się „czasem ekranowym" lub prościej RAMK.

Pisząc demo musimy pamiętać, aby wszystkie procedury (np. scroll, grajek do muzyki, bary) wykonywały się w jednym przebiegu nie dłużej niż jedną ramkę. Jeżeli ten warunek nie zostanie spełniony otrzymana animacja będzie złej jakości. Zapewne widzieliście czasem dema z wektorówką, która nie poruszała się płynnie - najprawdopodobniej było to wynikiem tego, iż procedury liczenia i rysownia wykonywały się w czasie większym niż jedna ramka. Złożoność niektórych dem sprawia, iż zmieszczenie się w jednym czasie ekranowym jest nie lada sztuką. Dlatego też często zdarza się, że celem dema jest pokazanie możliwości programisty, który podczas jednej ramki stawia 700 punktów lub animuje bryłę posiadającą 38 ścian. Pamiętajmy zatem, aby zawsze starać się pisać programy używając najszybszych rozkazów.

Następną ważną rzeczą niezbędną dla prawidłowego działania dema jest „ekranowe miejsce wykonywania procedury". Załóżmy, że chcemy aby w naszym demie na górze ekranu, między rastrami $20 a $70 kręciła się „wektorówka". Aby wszystko przebiegało poprawnie procedury rysowania wektorówki nie mogą być wykonywane w w/w zakresie rastrów. Wykorzystując rejestry VHPOSR i VPOSR miejsce rysowania możemy ustalić na np. $75 co pozwoli na uniknięcie kolizji. Natomiast w miejscu między rastrami $20 a $70 możemy zlecić wykonanie procedury grajka lub scrolla, którego umieścimy u dołu ekranu.

Przedstawię teraz schematyczny rysunek, który pokazuje jak Amiga ma zorganizowany ekran i jak zachowują się rejestry VPOSR oraz VHPOSR.

Górna część ekranu

rastry: $00 - $ff

Dolna część ekranu

rastry: $100 - $138

VPOSR - odczyt najbardziej znaczącego bitu pozycji pionowej.

Bit 0: skasowany dla rastrów $00-$ff
ustawiony dla rastrów $100-$138

VHPOSR - odczyt pozycji pionowej i poziomej.

Bity 15-8: pozycja pionowa - zmieniają się w granicach $00-$ff, $00-$38, $00-$ff,...
bity 7-0: pozycja pozioma.

Krzysztof „K.K." Kobus



Kącik początkującego kodera (cz. 5)

Przedstawiamy kolejną porcję mnemoników oraz procedur do badania czasu wykonywania rozkazów lub programów. Zanim jednak przystąpicie do lektury prezentowanej procedury proponuję zapoznać się z treścią artykułu z cyklu „Jak zrobić własne demo".

SWAP Dx „Swap Register Halves" - zmiana połówek rejestru.

Instrukcja ta zamienia ze sobą dwa słowa rejestru danych. Młodsze 16 bitów zostanie zamienione ze starszymi szesnastu bitami.

Znaczniki: X - nie zmieniany
N - jest kopią najstarszego bitu operandu
Z - zerowany jeśli wszystkie bity operandu są skasowane
V,C - zawsze zerowane

Przykład:

```
move.l     #$234f568a,d7
swap d7 -  w wyniku działania tych dwóch rozkazów
           w rejestrze d7 otrzymamy wartość $568a234f.
```

TST adres efektywny „Test an Operand" - testowanie operandu.

Za pomocą tej instrukcji możemy przetestować operand nie zmieniając jego wartości. Wynikiem działania tej operacji jest odpowiednie ustawienie znaczników. Testowany operand może mieć wielkość bajtu, słowa lub długiego słowa.

Znaczniki: X - nie zmieniany
N - ustawiany, gdy testowana liczba jest ujemna
Z - ustawiany, gdy wszystkie bity operandu są skasowane
V,C - zawsze zerowane

Przykład:

tst.b $40000 - znaczniki zostaną zmodyfikowane w zależności od bajtu znajdującego się pod adresem $40000.

JSR adres efektywny „Jump to Subroutine" - skok do podprogramu.

Za pomocą tej instrukcji możemy skoczyć do podprogramu, który znajduje się pod adresem określonym adresem efektywnym. „JSR" przed skokiem do podprogramu umieści adres następującej po niej instrukcji na stosie, aby po napotkaniu instrukcji powrotu (patrz RTS) zmodyfikować PC i powrócić do programu głównego. Mechanizm podprogramów jest bardzo wygodny i bardzo często wykorzystywany przez koderów, szczególnie przy tworzeniu programów użytkowych. Pozwala on na zaoszczędzenie pamięci oraz czyni sam program bardziej przejrzystym. Swoistym przykładem wykorzystania podprogramów są biblioteki (ang: libraries) znajdujące się w pamięci ROM lub na dyskietkach. Do nich to właśnie skaczemy za pomocą JSR gdy chcemy np. otworzyć okienko lub zapisać dane na dyskietce. Podprogramy te wykonują określone operację, po czym wracają do głównego programu zwracając w rejestrach lub strukturach potrzebne dane lub status.

Znaczniki: nie są modyfikowane.

Przykład:

jsr $70000 - skoczy do podprogramu, którego początek znajduje się pod adresem $70000

jsr -408(a6) -skoczy do podprogramu znajdującego się pod adresem, który jest sumą przesunięcia (tutaj: -408) i zawartości rejestru a6.

BSR przesunięcie „Branch to Subroutine" - skok względny do podprogramu.

Instrukcja ta jest analogiczna do JSR, z tą jednak różnicą, że zamiast adresu efektywnego podajemy przesunięcie. Dzięki temu instrukcja ta zajmuje mniej miejsca w pamięci i wykonywana jest szybciej. Przez to jednak jej zasięg jest mniejszy, gdyż przesunięcie jest wartością zapisaną na 16-tu lub 8-miu bitach.

Pisząc program w edytorze assemblera (np. SEKA) nie musimy martwić się o obliczanie przesunięcia. Wpisujemy po prostu etykietę, a w procesie assemblacji przesunięcie zostanie obliczone automatycznie.

Znaczniki: nie są modyfikowane.

Przykład:

bsr.s drukuj - wykona skok do podprogramu „drukuj". Przesunięcie zostanie zapisane na 8-miu bitach.
bsr.l save - skoczy do procedury „save". Przesunięcie zostanie wygenerowane jako 16-bitowe.

RTS „Return from Subroutine" - powrót z podprogramu.

Instrukcja ta zdejmie ze stosu adres do PC. Za pomocą RTS możemy wracać z podprogramów wywołanych przez BSR lub JSR do instrukcji znajdującej się bezpośrednio za nimi.

Znaczniki: nie są modyfikowane.

Przykład:

RTS

JMP adres efektywny „Jump" - skocz.

Wykonanie tej instrukcji powoduje skok czyli przepisanie do PC wartości określonej adresem efektywnym. Programista powinien pamiętać aby w programie umieszczać jak najmniej takich skoków,o ile to jest możliwe, zastępując je rozkazami JSR. Przy krótkich skokach - w obrębie jednego programu proponuję używanie instrukcji BRA, która wykonywana jest szybciej i zajmuje mniej miejsca w pamięci. Wszystkie te zabiegi sprawią, że nasz program będzie krótszy, szybszy oraz bardziej przejrzysty.

Znaczniki: nie są modyfikowane.

Przykład:

jmp $34568 - skoczy pod adres $34568
jmp Cont - skoczy do etykiety „Cont"

MULU adres efektywny, Dx „Multiply Unsigned"-mnożenie bez znaku.

Instrukcja ta mnoży 16-bitowe liczby określone adresem efektywnym i rejestrem danych, dając w efekcie 32-bitowy wynik, który zostaje pozostawiony w rejestrze danych. Mnożenie wykonywane jest w arytmetyce bez znaku. Programista powinien wiedzieć, że zarówno rozkazy mnożenia jak i dzielenia wykonywane są bardzo długo. Dlatego też powinien starać się zastępować je dodawaniem, odejmowaniem lub tablicami. Ta ostatnia metoda jest bardzo pamięciochłonna, lecz daje najlepsze rezultaty.

Znaczniki: X - nie zmieniamy

N - ustawiany, gdy najstarszy bit wyniku jest ustawiony
Z - Ustawiany, gdy wynikiem jest zero
V,C - Zawsze zerowane

Przykład:

mulu d0,d1 - młodsze słowa rejestrów d0 i d1 zostaną pomnożone, a 32-bitowy wynik zostanie umieszczony w rejestrze d1.

mulu #4,d7 - rejestr d7 zostanie pomnożony przez 4. Tą instrukcję można zastąpić sekwencją

```
add.l d7,d7
add.l d7,d7
```

która wykonana zostanie znacznie szybciej, a w rezultacie otrzymamy to samo.

MULS adres efektywny,Dx „Multiply Signed" - mnożenie ze znakiem.

Instrukcja ta ma działanie podobne do MULU, z tą jednak różnicą, że mnożenie odbywa się w arytmetyce ze znakiem.

Znaczniki: analogicznie jak MULU.

Przykład:

muls (a0),d0 -16-bitowa liczba znajdująca się pod adresem wskazywanym przez a0 zostanie pomnożona przez młodsze słowo rejestru d0. 32-bitowy wynik zostanie umieszczony w d0.

Przedstawiona dzisiaj procedura nawiązuje do artykułu „Jak zrobić własne demo". Służy ona do sprawdzania długości czasu wykonywania instrukcji. Pozwoli to na używanie w swoich programach (w szczególności w demach), rozkazów, które wykonują się szybciej. Przez to Wasze programy staną się szybsze i efektywniejsze.

Działanie tej procedury jest bardzo proste i składa się z trzech etapów:

1. Inicjacja i czekanie na raster;
2. Pętla czyli cykliczne powtarzanie badanej sekwencji rozkazów;
3. Obliczanie ilości rastrów i powrót.

Po przyciśnięciu lewego przycisku myszy wykonywnie procedury zostanie przerwane.

W wyniku działania tej procedury uzyskamy orientacyjną ilość rastrów potrzebnych na wykonanie zadanej sekwencji rozkazów, umieszczoną w rejestrze d6. Niestety w pewnych przypadkach mogą nastąpić przekłamania i uzyskany wynik będzie fałszywy. Spowodowane to może być tym, że czas potrzebny na wykonanie pętli będzie dłuższy niż jedna ramka lub wykonywanie przeciągnie się do dolnej połówki ekranu. Należy wtedy skorygować ilość powtórzeń w pętli (obecnie wartość 300).

A oto procedura:

```
WaitForRaster:
    clr.l    d6              ;Czyścimy rejestr d6, który
                             ;wykorzystamy później
    cmp.b    #$70,$dff006
    bne      WaitForRaster   ;Zaczynamy od rastra $70
    move.w   #300,d7         ;Ilość powtórzeń
    move.w   #$000f,$dff180  ;Włączamy kolor, który będzie
                             ;odzwierciedlał długość
                             ;wykonywania pętli
Loop:
    add.l    d3,d3           ;Tu umieszczamy sekwencję
                             ;rozkazów której długości
                             ;wykonywania chcemy badać
    dbf      d7,Loop         ;Jeżeli d7 różne od -1 to
                             ;skok do „Loop"
    move.w   #$0000,$dff180  ;Wyłączamy kolor bo pętla
                             ;zakończona
    move.b   $dff006,d6      ;Numer aktualnego rastra
                             ;do d6
    sub.b    #$70,d6         ;Odejmujemy numer początkowego
                             ;rastra, w ten sposób obliczamy czas
                             ;wykonywania pętli wyrażony w rastrach
    btst     #6,$bfe001      ;Jeżeli lewy przycisk myszy
                             ;przyciśnięty to
    bne      WaitForRaster   ;powrót
    rts
```

W przedstawionym wyżej przypadku badanym rozkazem jest „add.l d3,d3". Jest on równoważny rozkazowi „muls #2,d3". Proponuję teraz wstawić ten drugi w miejsce pierwszego i sami przekonacie się o tym, który wykonywany jest dłużej. Jeżeli będziemy chcieli zbadać długość wykonywania gotowej, własnej procedury wystarczy w miejsce za etykietą „Loop" wstawić „jsr NaszaProcedura". Podczas analizy problemem może być rozkaz „dbf d7,Loop", ponieważ jeszcze go nie opisywałem. Dokładnie zapoznacie się z nim w jednym z kolejnych odcinków z tego cyklu. Teraz wspomnę tylko, iż zmniejsza on o 1 młodsze słowo rejestru d7 i wykonuje skok jeżeli jego wartość jest różna od -1.

Krzysztof „K.K." Kobus

CIEKAWE ???

Przypadkowo odkryłem dosyć ciekawą cechę procesora Motorolla 68000. Procesor ten mimo 16 bitowej architektury zachowuje się dosyć dziwnie... a mianowicie gdy wykonuje program umieszczony w pamięci to pobiera nie kolejne rozkazy lecz dwa słowa jednocześnie.

Z tej cechy mogą wynikać dosyć ciekawe efekty. Oto przykład - program, którego celem ma być „ganianie" po pamięci (z cyklu „gdy nudzi się programiście").

```
         lea rozkaz (pc), a)
         lea kopia (pc), a1
rozkaz:  move.w (a0)+,(a1)+ ;rozkaz o długości 16 bitów
kopia:   rts
```

Program powinien kopiować sam siebie i skakać do kolejnego rozkazu, który znowu skopiuje siebie itd, itd.

Uruchamiając program stwierdzimy, że nie wykonuje się on w nieskończoność, ale powraca już w komórce o adresie „kopia" gdy wykona rozkaz rts. Można by przypuszczać, że programista popełnił jakiś błąd, ale oglądając pamięć od adresu „kopia" zauważmy, że jest tam rozkaz „move.w (a0)+,(a1)+"!!! Czy nie jest to ciekawe???

Ogłaszamy nieustający konkurs na programy najciekawiej wykorzystujące tą właściwość procesora MC68000.

Marcin „Duddie" Dudar

OD REDAKCJI

Drodzy Czytelnicy!

Poważna awaria techniczna - jaka miała miejsce w naszej drukarni - spowodowała znaczne opóźnienie druku lipcowego numeru. Obawialiśmy się, że niektóre oddziały Ruchu automatycznie zwrócą cały nakład, jako przeterminowany, nie przekazując go do kiosków. Pozbawiło by to Was możliwości nabycia naszego pisma. Zdecydowaliśmy się więc (w ostatniej chwili) na wydanie podwójnego numeru - co pozwoli również na zmiejszenie opóźnienia wydania następnych.

Nietypowa „konstrukcja" numeru podwójnego wynika z faktu, że część stron dla całego nakładu nasza drukarnia wydrukowała przed awarią i musieliśmy to uwzględnić przy składaniu numeru podwójnego.

Za wszystkie te anomalia serdecznie przepraszamy naszych Czytelników, mając nadzieję, że na przyszłość uda się takich sytuacji uniknąć.

Redakcja

spisu treści c.d. :

Poszukuję instrukcji programu X-CAD PROFESIONAL. Kwiatkowski, ul. Piaskowa 13a/1, 65-204 Zielona Góra.

Amiga MAUSE TURBO (nowa) - sprzedam lub zamienię na myszkę do C-64. Jarosław Skłodowski, ul. Świerczewskiego 3/2, 11-015 Olsztynek, tel. 192-855.

Proszę o pomoc: C-64 - drukarka D100 MPC, Ryszard Somka, 53-312 Wrocław, Drukarska 20A/9.

Poszukuję assemblerów, monitorów i programów użytkowych na C-64 z magnetofonem. Wymienię programy na C-64. Paweł Witek, Karłowicza 45/55, 58-506 Jelenia Góra.

POMOCY HACKERZY!!! Chcę napisać demo i nauczyć się assemblera C-64. Darek Matynia, ul. Wronia 47/65, 97-300 Piotrków Tryb.

Kupię Amigę 500. Oferty z ceną kierować: Artur Bujniewicz, ul. Mickiewicza 6/3, 78-520 Złocieniec.

Sprzedam nowy C-64 z magnetofonem (GWARANCJA!) - 1.700.000zł. Sławomir Leszczyński, ul. Krasickiego 14/31, 87-100 Toruń.

AMIGA - najlepsze gry, programy użytkowe - wysyłka pocztą. Ekspresowe terminy, katalogi gratis. Dla sklepówki rachunki. SOFTSTUDIO, Os. Tysiąclecia 54/6, 31-610 KRAKÓW, tel. 48-51-50.

Pomóżcie w obsłudze Sound Samplera do Amigi; w zamian służę oprogramowaniem. Przemysław Jeziorski, ul. Kolejowa 7/1, 10-284 Olsztyn.

Sprzedam nowe C-64 II, 1541 II, 1530-C2N, cartridge, programy, literaturę. Gdynia, tel 290-580.

PUBLIC DOMAIN PACK

PUBLIC DOMAIN PACK C-64

Marzec

Strona A
- FONT GRUB 1.0
- PROJEKTANT DUSZKÓW
- STRZAŁKA 64+
- PIRATEK - GRA
- V4.0 - SYMPHONIES
- CRUISER
- THE FIRST
- COMMERCIAL BREAK
- RELOKATOR 64
- KOREKTOR 64
- FLASH

Strona B
- HOT SHOT nr. 9 (zachodni magazyn fanów)
- BAD NEWS nr. 2 - j.w.
- DEMO - rekord - 290 sprite'ów!
- DEMO: NEW INTRO
- DEMO: LET'S DYCP
- KONTAKT CORNER - adresy, kontakty
- NEW FAST - działa z 1541 i 1541 II
- CSLINKER V2.0

Kwiecień

Strona A
- Digi - Organizer - program do tworzenia muzyki z użyciem digitalizacji dźwięku.

Strona B
- „ONE YEAR - RADIUS" - mega demo grupy RADIUS. Bardzo ładna grafika.

Maj

Strona A
- CRUEL SOLIDERS - demo
- DESTINATION'91 - demo
- SUCKER DJ! - demo (digi mix)
- MUSIC SEARCHER - do wycinania ilustracji muzycznych z programów

Strona B
- MEGA DEMO „INFOSYSTEM 91"

Czerwiec

strona A
- FONTEDITOR
- SINDATA EDITOR
- COLOR EDITOR
- DISK - NOTER
- GWIAZDY - demo graficzne
- FILGRAEPH 2.2/BML
- NOTE TO FLI V 2.2
- AFLI - EDITOR V 1.2
- RESET - MON,8,1
- TURBO - ASS 5
- ...HIGHLIFE #5
- AXEL NEWS #1
- DISK NOTKA/PADUA

strona B
- PSC - MAG #9'06/91
- CONSPIRE/OREGON - demo
- CONTACT DEMO/ORE
- SHOWPIX

PUBLIC DOMAIN PACK AMIGA

Marzec

- Najnowszy i najlepszy program muzyczny PROTRACKER V1.0 (pakiet programowy)
- Najlepsze muzyczki: NOW WAT? - DR. AWESOME
- AMOS - procedury
- DEMO grupy REBELS „TOTAL TRIPLE TROUBLE"

Kwiecień

- RUBBER VECTORS - demo
- KEFTALES - demo
- DISK MASTER V3.0
- Moduły muzyczne:
  - TECHNOSTYLE 2
  - GALAXY 2
- GRAFIKA - prezentujemy rysunki
  - RICK PARKS

Maj

- VIRUS X 5.0
- VIRUS TERMINATOR
- PARADOX - demo
- STORMCHILD - demo
- Moduły muzyczne:
  - MIAMI VOICE
  - ANTI ATARI SONG

Czerwiec

- POWER BOOT - własne menu dysku
- DISK CODING SYSTEM - program do zabezpieczania dysków
- Konwerter IFF - ANSI
- AUER NATION - demo
- Moduły muzyczne
- DOCS - opis gry ELWIRA
- LAMER DEFENCE - do wykrywania i niszczenia wirusów
- REWENG GO OF THE LAMER - grafika w trybie D_HAM

Spis treści zestawów z grudnia i stycznia patrz nr poprzedni naszego pisma.

Zestawy „64 plus 4 PUBLIC DOMAIN PACK" można zamawiać wpłacając na konto: Bank PKO SA Oddział w Bydgoszczy konto nr:5.09011-400522.7-136-11-111.0 następujących kwot: 20.000zł za pojedyńczy zestaw dla C-64, 25.000zł za zestaw dla AMIGI. Kwoty te obejmują koszt dyskietki, koszty kopiowania, opakowania i przesyłki pocztowej. Blankiety wpłat powinny być **CZYTELNIE** wypełnione i zawierać: **Imię i nazwisko, dokładny adres zamawiającego, skrót „PDP-64"**(jeśli zamawiamy zestaw dla C-64) lub **„PDP-A** "(zestaw dla Amigi). W prenumeracie zestawy kosztują: PDP-64 - 18.000zł (12 numerów 216tys zł), PDP-A - 22.000zł (12 numerów 264tys zł). Prenumeratę można zawrzeć w dowolnym terminie na okres od 3 do 12 miesięcy (do końca roku kalendarzowego). Prenumerata może obejmować miesiące od początku roku - tzn. zamawiając całoroczną prenumeratę np. w lipcu, w pierwszej przesyłce otrzymacie wszystkie poprzednie zestawy (tj. od stycznia, do lipca).

ZAMÓW NIE ZWLEKAJ!